SZCZEPANOWSKI

MICRO APPLICATION

9

AMSTRAD

PEEKS ET POKES DU CPC



UN LIVRE DATA BECKER

SZCZEPANOWSKI

MICRO APPLICATION

9

AMSTRAD

PEEKS ET POKES DU CPC

UN LIVRE DATA BECKER

Distribué par MICRO APPLICATION
147 Av. Paul Doumer
92500 RUEIL-MALMAISON

et également

EDITIONS RADIO
3 rue de l'Eperon
75006 PARIS

ISBN 2-86899-017-7

Traduction Française et mise en pages assurées par Alexandre UNGERER

Edité par Frédérique BEAUDONNET
Léo BRITAN
Philippe OLIVIER

REMARQUE IMPORTANTE !

Les branchements, procédés et programmes reproduits dans ce livre sont communiqués en dépit de l'existence éventuelle de brevets d'invention. Ils sont destinés à l'usage amateur et éducatif et ne peuvent en aucun cas être employés à des fins professionnelles.

L'auteur a apporté le plus grand soin à l'élaboration de l'ensemble de ce livre. Bien que leur reproduction ait été réalisée en faisant appel à des moyens de contrôle efficaces, des erreurs éventuelles ne sont pas exclues tout à fait. C'est pourquoi la société MICRO APPLICATION tient à signaler que toutes les indications sont données sans garantie et qu'elle décline toute responsabilité quant aux conséquences que peuvent entrainer des indications erronées. Nous remercions les aimables lecteurs de nous signaler les erreurs éventuelles.

- NDE :

Dans les listings de programmes la lettre p a pour signification l'élévation à la puissance qui lors de la frappe du programme est ↑

SOMMAIRE

PREFACE

Vous êtes vous déjà demandé comment fonctionne votre CPC 464 ? Avez-vous toujours voulu savoir ce qu'est un système d'exploitation ? Les guides de l'utilisateur, aussi détaillés soient-ils, n'offrent malheureusement pas de réponses totalement satisfaisantes à ces questions. Beaucoup de trucs et astuces, qui facilitent pourtant grandement la vie du programmeur, ne se dévoilent qu'après un regard derrière les coulisses. C'est pourquoi ce livre est là pour vous expliquer comment fonctionne votre ordinateur, quel processus se déclenche lorsque vous appuyez sur la touche ENTER, etc.

Le titre PEEKS et POKES pourrait laisser à penser qu'il s'agit ici d'un livre ne traitant que de ces deux instructions. Heureusement il n'en est rien (sinon je me serais d'ailleurs moi-même beaucoup ennuyé). Le titre a été calqué sur le précurseur de ce livre: "PEEKS et POKES du Commodore 64" dont il reprend la conception et la présentation. D'ailleurs pas plus un CBM 64 qu'un Amstrad ne se programme uniquement avec des PEEKS et des POKES. Par contre nos deux instructions ont la bonne réputation de permettre une introduction facile au système d'exploitation et au langage machine. Et c'est exactement ce que nous voulons obtenir ici.

Accompagnant toutes ces explications, de nombreux trucs vous seront présentés, avec en plus un petit extra: à la fin de chaque chapitre un récapitulatif de tous les "trucs" vous sera proposé. Vous pourrez ainsi ultérieurement les retrouver très rapidement sans être obligé de relire tous les détails du texte. Pour vous faciliter le travail, les mots répertoriés dans l'index, à la fin du livre, apparaîtront soulignés dans le texte.

Nous nous attaquerons également au bien maigre vocabulaire d'instructions graphiques du CPC, que nous étendrons, et ceci de manière accessible à tous. Il n'est pas nécessaire d'avoir fait des études poussées d'informatique pour comprendre tout ce qui est écrit dans ce livre. Et si malgré tout le langage machine vous semble encore du chinois, vous trouverez un chapitre que je vous conseille de consulter aussi souvent que nécessaire et qui vous permettra de faire vos premiers pas dans ce langage. Libre à vous ensuite de décider si vous voulez en rester à l'assembleur ou si vous désirez vous attaquer à d'autres langages de programmation, peut-être PASCAL ?

Il ne me reste plus qu'à vous souhaiter beaucoup de plaisir à la lecture de ce livre, ainsi que beaucoup de succès aux "expérimentations" avec votre CPC 464.

Hans Joachim Liesert
Münster, novembre 1984

1.COMMENT FONCTIONNE UN ORDINATEUR ?

Dans ce qui suit vous allez faire plus ample connaissance avec votre Amstrad et son fonctionnement. Ceux d'entre vous qui se sentent à l'aise dans les notions de bases de l'informatique peuvent sauter ce passage sans regret. Que les "supercracs" parmi vous me pardonnent, pour rester clair j'ai volontairement simplifié certaines notions.

1.1. UN ORDINATEUR AUSSI A DES BUS

Voyons d'abord quelques généralités. Tout ordinateur peut adresser un certain espace mémoire, c.à.d. un certain nombre de cases mémoire ou octets (en anglais byte). Cette quantité dépend du nombre de lignes d'adresses que possède le microprocesseur. Chaque ligne d'adresse représente un bit (j'espère que vous savez ce qu'est un bit, sinon allez vite consulter votre Guide de l'Utilisateur du CPC) et peut donc prendre deux états: 0 et 1.
Le microprocesseur Z-80, le cerveau de votre Amstrad, possède 16 lignes d'adresses qui forment ensembles ce que l'on appelle le BUS d'adresses. Avec ces 16 lignes il peut adresser 2p16 = 65535 = 64K (Kilo-octets).

N.B.: dans toute la suite nous utiliserons le caractère "p" comme signe "élevé à la puissance".

ex: 2p3 = 2 puissance 3 = 2*2*2 = 8

Peut-être avez vous déjà remarqué que le CPC possède 64K de RAM + 32K

de ROM = 96K, ce qui est bien plus que les 64K adressables ? Nous verrons plus tard comment cela peut quand même fonctionner.
En plus du BUS d'adresses, votre Z-80 possède encore deux autre BUS. Tout d'abord le BUS de commande. Ceci n'est en fait que
le nom que l'on donné à l'ensemble des lignes de commande, comme par exemple celle qui détermine si on lit ou si on écrit dans la mémoire.
Le BUS de données est bien plus important. Il est
constitué de 8 lignes et a pour mission de transporter les données (en fait des octets: 8 bits = 1 octet) à l'intérieur de l'ordinateur.
BUS d'adresses et BUS de données sont tous deux reliés à tous les composants de votre ordinateur qui sont gérés par le microprocesseur, soit la RAM, la ROM et d'autres circuits.
A chaque fois que le microprocesseur exécute une instruction utilisant des données ne se trouvant pas dans le Z-80, il envoie d'abord une adresse sur le BUS d'adresse (comme un numéro de téléphone). Le circuit mémoire auquel de ces octets on veut s'adresser. Ensuite nous avons deux possibilité: soit le microprocesseur envoie sa donnée sur le BUS de données d'où la mémoire la récupère, soit au contraire c'est la mémoire qui envoie son "contenu" au Z-80, toujours par l'intermédiaire du ? BUS de données, bravo.

Tout ceci est valable pour n'importe quel microprocesseur à 8 bits (8 bits = largeur du BUS de données). Mais assez de généralités, il est temps de nous intéresser aux particularités de notre CPC 464.

1.2. CONFIGURATION "HARDWARE" DU CPC

Pas de panique, ceci reste abordable par le non-technicien. Mais pour

la compréhension des prochains chapitres, il vous sera fort utile de connaître un peu de la vie interne de votre CPC 464.
A la fin de ce paragraphe, vous trouverez une représentation très schématisée de l'intérieur de votre ordinateur (fig.1). Comme vous pouvez le voir ROM et RAM se recoupent partiellement, c'est à dire qu'ils occupent les mêmes adresses mémoire. Il y a la apparemment un problème, le BUS de données ne pouvant prendre deux valeurs différentes en même temps. Il doit donc y avoir un moyen de décider à qui on s'adresse: ROM ou RAM. Suivant les désirs du microprocesseur l'une ou l'autre mémoire sera déconnectée. Ceci peut fonctionner parce-que les espaces mémoire se recoupant sont réservés à des tâches différentes, qui ne nécessitent pas d'être effectuées simultanément.
A propos, avec BASIC vous n'avez accès qu'à la RAM, avec PEEK et POKE. Grâce à un petit truc, on peut quand même accéder à la ROM, je vous expliquerai plus loin comment procéder.

Une partie de la RAM est occupée par la RAM vidéo (mémoire écran). C'est là que le Circuit Intégré 6845 va chercher ses informations. Le rôle de ce CI est de transformer les données de la mémoire écran en signaux vidéo pour le moniteur.

A coté du 6845 il y a d'autres CI, comme par exemple le AY 3-8912 (non, non, ce n'est pas la peine d'apprendre cette référence par coeur) qui est responsable du son dans votre CPC, c.à.d. qu'à partir des instructions qu'il reçoit, il doit produire quelque chose d'audible.
Pour des raisons techniques, le AY 3-8912 n'est pas relié directement au Z-80, les données lui sont transmises par un CI interface: 8255 (voir fig.1).
Ce composant gère un certain nombre de périphériques comme le magnéto-cassette, les port d'expansion ou le clavier. Il est à noter

que le clavier, bien qu'intégré dans l'ordinateur, est considéré par le Z-80 comme un périphérique externe.
Si par exemple le Z-80 veut aller chercher un caractère au clavier, alors il le fait savoir au circuit interface. Lui de son coté va voir si une touche a été pressée. Dans l'affirmative l'AY 3-8912 envoie au Z-80 le code du caractère par l'intermédiaire du BUS de données.
Supposons maintenant qu'on veuille fournir un octet à un périphérique branché sur un port d'expansion. Le Z-80 transmet le caractère au 8255 qui à son tour l'enverra au périphérique (par exemple une imprimante) dès que celui-ci sera prêt.
Il y a enfin un dernier CI qui règle divers processus internes, mais qui est sans grande importance pour nous ici.

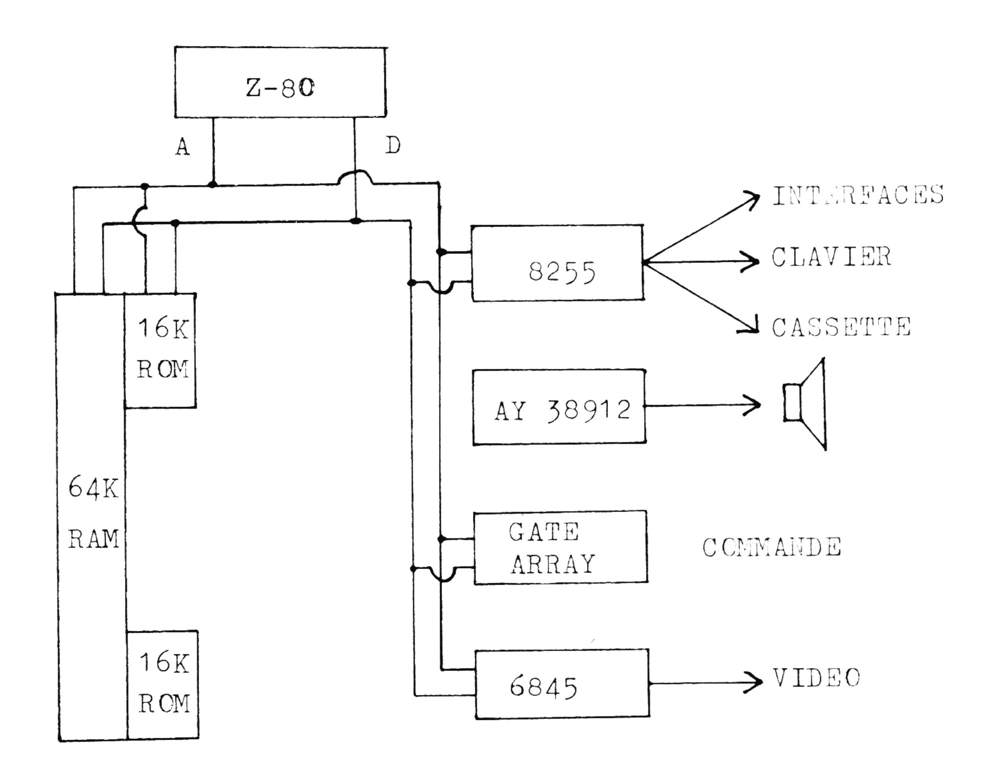


Fig. 1 Configuration du CPC 464

1.3. REPARTITION DE LA MEMOIRE

Nous avons vu au chapitre précédent que le CPC 464 possède 64K de RAM. Or seuls 42,5K sont disponibles au BASIC et vous vous demandez avec raison où sont passés les 21,5K manquant.

A première vue, il devrait être possible de réserver plus de mémoire au BASIC. Malheureusement l'ordinateur aussi se sert de la RAM. Citons en premier lieu la RAM vidéo. Son but est de représenter en mémoire ce que vous voyez à l'écran. A chaque fois que quelque chose doit se modifier à l'écran, le microprocesseur change la valeur correspondante dans la RAM vidéo. Le CI responsable de l'image lit à intervalles réguliers dans la mémoire écran pour savoir quels points il doit allumer. Cette répartition des tâches permet au microprocesseur de ne pas être surchargé inutilement.
La RAM vidéo nécessite 16K (qui se retranchent donc de la mémoire disponible en BASIC) et se trouve aux adresses 49152 jusqu'à 65535. Elle occupe donc le même emplacement que l'interpréteur BASIC.
Faisons un petit calcul: 64K - 16K = 48K. Où sont passés les 5,5K manquant ?

Tout microprocesseur se sert d'une certaine quantité d'octets, appelée Stack ou Pile, dans laquelle il entrepose ses données "personnelles". Par exemple, à chaque saut à un sous-programme, le Z-80 doit se souvenir d'où il a été appelé pour pouvoir y retourner à la fin de la routine. La pile permet au microprocesseur de s'y retrouver (voir aussi chap. 1.4.). Elle se trouve dans le domaine 48896 à 49151 et occupe ainsi exactement 256 octets. Je vous déconseille vivement de POKER dans ce domaine, vous perturberiez complètement le fonctionnement de l'ordinateur qui "décrocherait"

probablement. La seule issue possible serait alors d'éteindre et de rallumer votre CPC.
Pour pouvoir travailler à la fois avec la RAM et la ROM, il se trouve une routine dans la RAM (recopiée au préalable par le Z-80 de la ROM dans la ROM) qui permet de passer de l'une à l'autre mémoire. Encore d'autres routines sont entreposées dans la RAM, entre l'interpréteur BASIC et la mémoire écran. Par conséquent, gare aux résultats imprévisibles si vous modifiez des octets dans ce domaine à l'aide de l'instruction POKE. Dans la grande majorité des cas votre CPC prendra congé de vous et ne réagira plus du tout à vos implorations. Encore une fois, la seule solution sera d'éteindre le micro, et adieu vos programmes qui se trouvaient en mémoire !

Mais n'en restons pas là. Le système d'exploitation et l'interpréteur BASIC utilisent un certain nombre de cases mémoire pour stocker par exemple un résultat intermédiaire d'une opération arithmétique, des données à échanger entre deux programmes ou encore des caractères tapés au clavier.

Ceci m'amène à vous parler du tampon-clavier. Il vous autorise à taper des caractères sur votre clavier avant même que BASIC ne soit prêt à les recevoir. Vous pouvez d'ailleurs le tester vous-même. Tapez le tout petit programme qui suit, et démarrez-le avec RUN:

```
1 FOR i = 1 TO 10000: NEXT i
```

Pendant que le programme tourne, entrez quelques caractères au clavier. Tant que BASIC est occupé, rien ne se passe à l'écran. Mais une fois qu'il aura parcouru 10000 fois sa boucle, le message READY

s'affichera, suivi des lettres que vous avez tapées auparavant. Le système peut ainsi mémoriser jusqu'à 20 caractères. Vous pouvez donc, pendant que votre programme exécute des calculs compliqués et longs, déjà entrer le texte pour la prochaine instruction INPUT. Mais attention, un "break" efface la totalité du tampon-clavier !

Si vous désirez des informations supplémentaires quant à la répartition de la mémoire de votre CPC, veuillez vous reporter à l'annexe que vous trouverez en fin de livre. Un plan détaillé des adresses mémoire vous y est proposé.

1.4. POINTEURS ET PILE

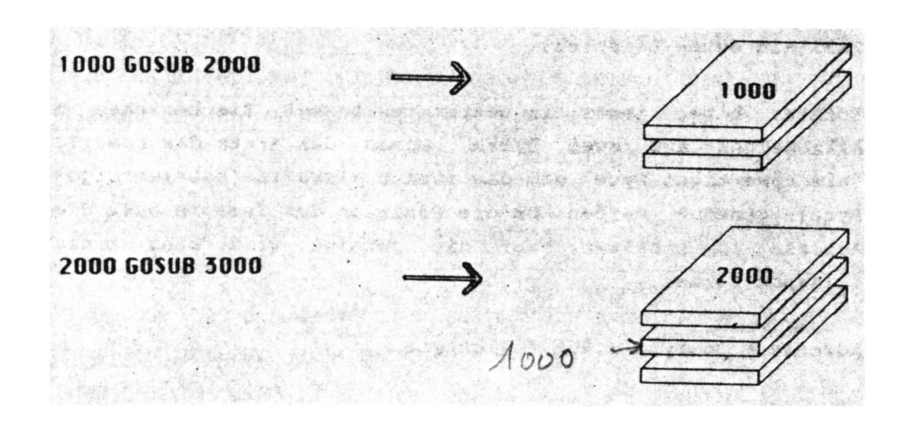
Pointeur et pile sont deux termes techniques que vous rencontrerez souvent.
Comme son nom l'indique, le pointeur (en anglais pointer), appelé aussi vecteur, pointe sur un certain endroit de la mémoire. Ainsi par exemple le pointeur-curseur indique le lieu de la RAM vidéo où se trouve le curseur et représente donc l'adresse à laquelle le prochain caractère devra être affiché.
Nous avons aussi des vecteurs qui pointent sur des sous-programmes, rendant ainsi les programmes plus flexibles. A l'aide d'un tableau de pointeurs de sous-programmes un programme donné peut par exemple sélectionner l'un ou l'autre des vecteurs, tout comme PRINT#8 choisit le huitième vecteur de la commande de l'imprimante (bien qu'en réalité l'instruction PRINT ne fonctionne pas tout à fait sur ce principe, c'est en tout cas un très bon exemple).

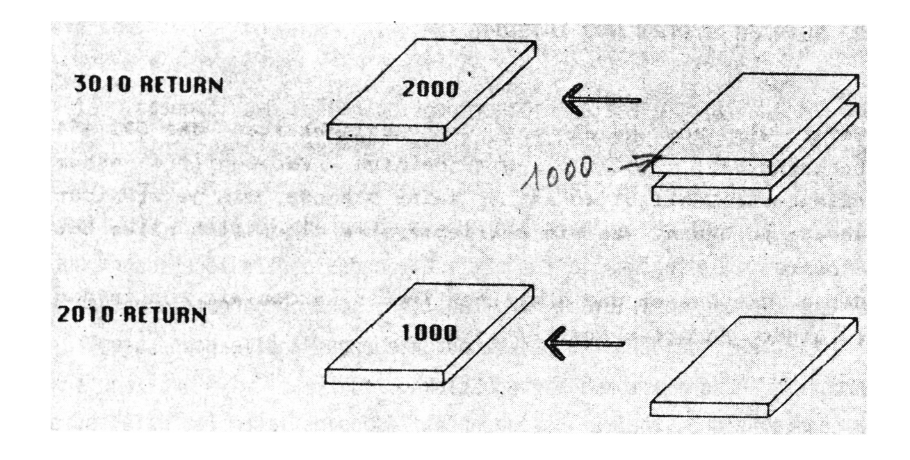
Un pointeur a toujours un certain format. En général il est constitué de deux octets, dont on appelle le premier lowbyte (octet de poids faible) et le second highbyte (octet de poids fort).
Pour obtenir la valeur de l'adresse pointée par le vecteur, on utilise la petite formule suivante:

adresse = lowbyte + 256 * highbyte

Si vous travaillez dans le système hexadécimal, c'est encore plus simple. Il suffit d'écrire côte à côte les deux valeurs, lowbyte à droite et highbyte à gauche, et vous obtenez la valeur hexadécimale de l'adresse.
Dans la mémoire de votre ordinateur, le lowbyte se trouve toujours avant le highbyte, cela fait partie des particularités d'un microprocesseur.
Les vecteurs à un seul octet s'utilisent de manière différente. Leur contenu (0-255) est additionné à une adresse de base fournie par le programme appelant.

Le rôle de la pile est de stocker provisoirement des données, et de les restituer ensuite dans l'ordre inverse, lorsque l'on en à besoin. La pile porte bien son nom, puisqu'on ne peut rajouter (resp. aller chercher) des données que sur le sommet de la pile. Nous avons déjà vu l'exemple classique d'utilisation de la pile lors d'un appel de sous-programme: avant de quitter le programme principal, l'ordinateur "dépose" l'adresse courante dans la pile pour savoir où repartir lorsqu'il tombe sur un RETURN.
A condition que toutes les données déposées sur la pile soient retirées à temps, il n'y aura pas de problème lors du retour en fin de sous-programme.


1000 GOSUB 2000
1000
2000 GOSUB 3000
2000
1000


3010 RETURN
2000
1000
2010 RETURN
1000

Le principe de la pile est le même pour le langage machine que pour BASIC. A ce propos sachez que dans les instructions du type FOR-NEXT la pile est également utilisée pour connaître l'adresse de retour après par exemple le NEXT. Voilà la raison pour laquelle sous-programmes et boucles ne doivent pas se recouper. En effet, dans l'exemple de la figure, si on sortait du sous-programme 3000 avant de sortir de 2000, alors le microprocesseur irait chercher la mauvaise valeur sur la pile (puisque 2000 se trouve au sommet) !

*****

EN BREF: **pointeur**

adresse = lowbyte + 256 * highbyte

lowbyte = adresse - INT(adresse/256)*256

highbyte = INT(adresse/255)

en général un pointeur (ou vecteur) est constitué de deux octets, représentés en mémoire dans l'ordre LOW/HIGH.

*****

2. SYSTEME D'EXPLOITATION ET INTERPRETEUR

Avant d'aller plus avant dans la découverte des possibilités du CPC 464, nous devrions nous familiariser un peu avec quelques termes techniques. Après tout il n'y a pas de honte à s'avouer que jusqu'à présent nous ne savions pas qu'elle est la fonction d'un système d'exploitation, n'est-ce pas ?
Ce chapitre est là pour nous familiariser avec ce genre de question.

2.1. UN AIDE TOUJOURS FIDELE

Aucun ordinateur ne peut s'en passer, vous connaissez déjà beaucoup de noms pour le désigner, mais peu de débutant savent de quoi il en retourne exactement- je veut parler du système d'exploitation.
Vous le savez, le système d'exploitation (en anglais OS: Operating System) est indispensable au fonctionnement de votre ordinateur. Par exemple pour envoyer un octet du Z-80 à l'imprimante, l'ordinateur se sert d'une multitude de programmes en langage machine qui se trouvent dans la ROM système d'exploitation. Bien que l'utilisateur n'ait pas besoin d'intervenir, quantité de tâches sont effectuées pour assurer un déroulement sans problème des opérations.
Nous pouvons donc dire que le système d'exploitation coordonne et assure la bonne entente entre les différents périphériques (que ce soit le clavier, l'imprimante etc.). Pour chaque périphérique il existe une routine qui lui est propre, et qui peut être utilisée par BASIC ou par vos programmes à vous. Il vous suffira d'apprêter

correctement les données à transmettre, puis d'appeler la routine adéquate.

Vous avez certainement déjà entendu parler d'autres systèmes d'exploitation comme par exemple CP/M ou MS-DOS (pour ne parler que des plus connus) et aussi de "systèmes d'exploitation standards".
Ces systèmes ont été conçus de telle manière qu'ils n'aient pas besoin d'appeler des sous-programmes. Ils se contentent de déposer dans des registres spéciaux des données et des numéros d'instructions. Il est ainsi possible d'adapter un même système d'exploitation à différents ordinateurs, en gardant les mêmes codes. Je voudrais attirer ici votre attention sur un petit inconvénient du langage machine: la plupart du temps, lorsque l'on réécrit une routine en l.m., les adresses de début de programme ne sont plus les mêmes, tout simplement parce que le nouveau programme n'utilise pas nécessairement le même nombre d'octets que l'ancien. Cela veut dire que sur chaque ordinateur les routines ont des adresses différentes, et comme les routines s'appellent mutuellement, vous imaginez les complications. Les codes d'instruction permettent de palier à cet inconvénient, et ainsi les concepteurs de logiciels n'ont pas à modifier leurs programmes pour chaque machine.

2.2. L'INTERPRETEUR

Le nom de ce programme en langage machine nous renseigne déjà assez sur sa fonction: il s'agit de traduire ("interpréter") les instructions BASIC en suites d'actions exécutables par le Z-80. En effet le Z-80 ne comprend qu'un vocabulaire bien précis d'instruction

et qui est différent pour chaque microprocesseur. Au départ, une ligne de BASIC n'est qu'une suite de symboles qui ne l'intéresse guère. Ce n'est que lorsqu'il reçoit le caractère ENTER que l'interpréteur commence sa première phase de traduction
(pour les amateur de SF: état d'urgence No 1) !
Et là le brave microprocesseur fait des heures supplémentaires pour reconstituer, à partir des lettres et des chiffres qu'il rencontre, les numéros de lignes, les instructions BASIC et les données. A chaque instruction est affecté un code spécial, qui lors de l'exécution sollicitera BASIC à appeler le bon sous-programme.
A ce stade, la suite de symboles a été traduite en codes mais ne se trouve pas encore en mémoire. Il faut pour cela d'abord faire de la place pour l'insérer (dans le cas où elle a été ajoutée à un programme existant). Ceci fait, elle vient enfin s'intercaler dans la mémoire.
Aussi incroyable que cela puisse paraître, toute cette suite d'opérations est effectuées dans le très bref laps de temps qui sépare les instants où nous pressons la touche ENTER et le moment où le curseur réapparaît !

Lorsque vous entrez RUN, c'est la deuxième phase de la traduction qui commence. L'interpréteur va chercher les codes en mémoire les uns après les autres et les exécute. Il n'a plus besoin d'aller reconnaître par exemple la suite de lettre P-R-I-N-T, il lui suffit de sauter à l'adresse indiquée par le code. On gagne ainsi un temps considérable.

2.3. LES INTERRUPTIONS

Les interruptions, très mal vues entre gens de bonne société, sont au contraire de mise chez les ordinateurs. Car ce sont justement elles qui permettent des applications exceptionnelles qui sans ce petit détail ne seraient pas envisageables.
Aussi bien le système d'exploitation que l'interpréteur BASIC sont des programmes en langage machine. Tous deux sont automatiquement mis en route lorsque vous allumez l'ordinateur et tournent tant qu'un autre programme en l.m. n'a pas été appelé. Si cela s'est produit avec l'instruction CALL, l'ordinateur retournera à BASIC après l'exécution du programme.

Comme vous l'avez déjà vu en BASIC, un ordinateur ne peut faire tourner qu'un seul programme à la fois (exception faite des systèmes à multi-processeurs). Pourtant l'interpréteur et le système d'exploitation sont deux programmes bien distincts qui doivent parfois s'exécuter simultanément. Comment cela est-il possible ?

La méthode la plus simple pour exécuter deux programmes "presque" simultanément, est de provoquer des appels mutuels. A chaque fois que BASIC en a terminé avec une tâche, il appelle le système d'exploitation, et réciproquement. Cela se produit par exemple lorsqu'on s'adresse à un périphérique. Mais appliqué au clavier, cela voudrait dire que celui-ci ne peut être lu que lorsque c'est au tour du système d'exploitation de tourner. Pourtant nous aimerions qu'au moins la touche ESC puisse conserver sa fonction à tout moment pour pouvoir arrêter par exemple un programme qui boucle. Pour résoudre ce problème, les concepteurs d'ordinateurs inventèrent l'INTERRUPTION. Toutes les 1/50 secondes, le microprocesseur interrompt le programme

en cours, quel qu'il soit (BASIC, SE ou programme personnel), et saute dans un sous-programme spécial, réalisant entre autres la lecture-clavier. S'il reconnaît que la touche ESC a été
tapée, il sortira du programme BASIC en cours. Si c'est une autre touche qui a été frappée, il la stockera dans le tampon-clavier.
Vu du côté de l'utilisateur, il semble que le clavier soit lu en permanence, car même un as de la frappe ne dépassera guère les 15 caractères par seconde. Par contre pour l'ordinateur l'espace entre deux interruptions paraît une éternité, puisqu'il accomplit environ 4.000.000 cycles par seconde (!) et qu'une instruction machine nécessite en moyenne 8 à 10 de ces cycles. Il peut donc exécuter des milliers d'instructions avant qu'une interruption ne vienne le déranger dans son labeur. Après l'exécution du programme d'interruption, il reprend à l'endroit précis où il avait quitté son travail.

Il existe certaines instructions pendant lesquelles les interruptions ne sont pas possibles, il en est ainsi lors de certaines opérations avec la cassette. On s'en aperçoit quand, lors d'un chargement, les couleurs s'arrêtent de clignoter, car elles aussi sont commandées par des interruptions. On change simplement de couleur à chaque appel d'interruption.

Vous-même pouvez provoquer des interruptions en BASIC en utilisant les instructions AFTER et EVERY qui vous permettent d'écrire vos propres routines d'interruptions. Le principe est exactement le même qu'en langage machine.
Lorsque l'interpréteur tombe sur AFTER ou EVERY, il bloque un TIMER, qui déclenchera après un certain temps, comme un réveil,
une sonn- pardon une interruption. A ce moment BASIC laisse tout en plan et va s'occuper de la routine d'interruption.

Une demande d'interruption peut être faite par d'autres composants de l'ordinateur pour signaler par exemple qu'une donnée est prête à être reçue sur le port d'expansion. L'interpréteur, lui, ne réagit qu'aux signaux du TIMER.

Le RESET représente une forme très particulière d'interruption. Il force le Z-80, tout comme lors de la mise sous tension, à retourner au programme situé à l'adresse 0, la routine d'initialisation. Cette routine vide la mémoire et met l'ordinateur en configuration de départ. Nous voyons donc qu'en plus du classique SHIFT-CTRL-ESC nous pouvons provoquer un RESET à partir de BASIC avec CALL 0, mais attention, toutes les précieuses données que vous aviez calculées avec amour s'évaporent ! Pensez à les sauvegarder sur cassette avant une opération aussi osée.

2.4. POUR CELUI QUI EN VEUT PLUS: DES EFFETS SPECIAUX

Imaginez-vous dans la situation suivante: Vous avez trouvé dans une des nombreuses revues d'informatique, un super programme à taper. Vous avez réussi, au bout d'un long chemin de Croix, à rentrer les 20K de listing dans le ventre de votre ordinateur, mais horreur, alors que plein d'espoir vous tentez l'exécution, le message tant redouté s'affiche sous vos yeux: ERROR ...
Il n'y a rien à faire, si vous ne voulez pas être obligé de comparer votre listing lettre pour lettre avec l'original, il va falloir essayer de comprendre le fonctionnement du programme. Si seulement il n'y avait pas tous ces POKES auxquels on ne comprend rien ! Il est grand temps de lever le voile sur le pseudo-mystère qui entoure ces

instructions.

2.4.1. PEEK & POKE

Prenons tout d'abord l'instruction POKE. Je vous rappelle la syntaxe: POKE adresse,octet. L'adresse peut prendre les valeurs 0-65535, et l'octet les valeurs 0-255. Cette instruction met en mémoire l'octet donné, à l'adresse donnée. Suivant l'adresse, on peut ainsi remplir l'écran, écrire une instruction machine ou encore beaucoup d'autres choses. Nous pouvons aussi "duper" le système d'exploitation et l'interpréteur en venant interférer dans leurs données. Tenez, justement, pour connaître ces données, nous allons utiliser l'instruction PEEK. Je pense que vous savez comment elle fonctionne: PRINT PEEK(adresse) affiche l'octet se trouvant à l'adresse mentionnée.

Notez bien que PEEK est une FONCTION, par conséquent on ne peut l'utiliser que dans une affectation (A=PEEK...) ou une autre expression.

PEEK et POKE ont en commun le fait qu'ils concernent tous deux une adresse. C'est pourquoi il est conseillé, lorsque l'on tombe sur une telle instruction dans un programme, de vérifier dans quel domaine de la mémoire on se trouve. La plupart du temps on en déduit assez facilement leur effet.

2.4.2. CALL

Nous arrivons là à une instruction qui ne devrait intéresser que le programmeur en langage machine: CALL adresse, sert à appeler un programme l.m. se trouvant en mémoire à partir de l'adresse donnée.
Une fois la routine exécutée, l'interpréteur revient au programme BASIC qui se poursuit juste après le CALL.
Notons, pour ceux que cela intéresse, que l'on peut transmettre des données au programme l.m. en les écrivant, séparés par exemple par une ",", juste après l'adresse du CALL.

2.4.3. PETITE EXCURSION AU PAYS DU CALCUL BINAIRE

Avec les instructions que nous venons de voir, on en trouve d'autres dont vous ne connaissez peut-être pas encore toutes les finesses.
Voyons d'abord AND, OR, XOR et NOT. Peut-être ne les avez vous jamais utilisées, ou alors dans des constructions IF-THEN du style:

```
IF A=0 AND B=0 THEN 100
```

Pourtant leur vocation d'origine était d'établir des combinaisons logiques entre variables et nombres. Il faut savoir que votre ordinateur traite les comparaisons comme des nombres. Pour vous en assurer, essayez donc ce qui suit:

```
PRINT(1=2)
PRINT(1=1)
```

Si une comparaison donne un résultat vrai, alors on obtient -1, un résultat faux donne 0. Dans le système binaire de notre ordinateur, -1

est représenté comme ceci: 1111 1111. Si on considère ce nombre comme non-signé, cela donne 255. Mais qu'est-ce que cela a à voir avec notre instruction BASIC ?
Un programme sortira toujours d'une construction IF-THEN lorsque la comparaison donne 0. Par conséquent la ligne suivante est tout à fait raisonnable:

```
IF 3*A THEN 100
```

Tout repose sur une combinaison de comparaisons dont le résultat détermine la suite du déroulement. Pour bien comprendre le fonctionnement de cette logique, nous allons faire une petite excursion au pays du calcul binaire.

AND, OR, XOR et NOT sont appelés OPERATEURS BOOLEENS, et servent à combiner des états logiques. Comme vous le savez, il est très commode de représenter un état logique à l'aide d'un bit (0 pour "faux", 1 pour "vrai").
On compare toujours deux bits. Le résultat des différentes opérations vous est donné dans les tableaux suivant:

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

Avec OR le résultat vaut 1 lorsque bit 1 OU bit 2 est à 1, alors que pour AND il faut que bit 1 ET bit 2 soient à 1.
Le OU exclusif (XOR) fournira un résultat vrai si l'une ou

l'autre des entrées, mais pas les deux, est vraie.
NOT est plus simple, il n'a qu'une seule entrée qui ressort inversée:

```
+-----+---+---+
I XOR I 0 I 1 I
+-----+---+---+      +-----+---+---+
I  0  I 0 I 1 I      I NOT I 0 I 1 I
+-----+---+---+      +-----+---+---+
I  1  I 1 I 0 I      I     I 1 I 0 I
+-----+---+---+      +-----+---+---+
```

Jusque là tout va bien. Malheureusement des bits isolés comme nous venons de les voir ne nous servent pas à grand chose en BASIC. Nous avons à faire ici à des nombres décimaux. Pour connaître le résultat de 45 AND 123 nous allons procéder de la manière suivante:

1) Convertir le nombre décimal en binaire.
Reportez vous au guide de l'utilisateur de votre CPC pour savoir comment faire.

Voilà ce que donnent les nombres 45 et 123:

```
 45 = 00101101
123 = 01111011
```

2) Combiner bit par bit les deux nombres.
Dans le cas de nos 45 et 123 vous devez obtenir ceci:

```
    00101101
AND 01111011
```

= 00101001

3) Convertir ce résultat binaire en décimal.

00101001 = 41

Vous pouvez bien sûr aller beaucoup plus vite en entrant simplement PRINT 45 AND 123, mais ce que nous venons de faire a l'avantage d'être bien plus explicite.

Maintenant vous vous demandez avec raison à quoi tout cela peut bien servir. En plus des comparaisons, ces opérateurs sont utilisés à chaque fois que l'on veut modifier ou tester quelques bits isolés dans un octet. Ainsi en appliquant AND 254 à un octet, vous êtes sûr de mettre a 0 le bit le plus à droite, de même que OR 1 réalise l'opération inverse, c.à.d. que dans tous les cas il met le bit le plus à droite à 1.

2.4.4. INSTRUCTIONS POUR LES PORTS

C'est par l'intermédiaire des connecteurs qui se trouvent derrière votre CPC que vous pouvez communiquer avec les périphériques. Il existe des commandes spéciales qui vous permettent d'accomplir ceci en BASIC. INP(port) va chercher un octet sur le port indiqué entre parenthèses. Remarquez que le numéro du port n'a rien à voir avec une adresse mémoire, il est lié à un système d'adressage dont la compréhension nécessite déjà de sérieuses connaissances en langage machine. Je voudrais tout de même vous expliquer les différentes

instructions s'y rapportant, pour que vous compreniez au moins de quoi il en retourne.
Avec OUT port,octet vous pouvez envoyer un octet sur le port. Vous avez sans doute remarqué la grande similitude existant entre ces instructions et PEEK et POKE.
Il nous reste encore à voir une instruction bien mystérieuse: WAIT port,X,Y.Sa fonction est tout ce qu'il y a de plus désagréable aux yeux d'un ordinateur moyen; en effet, elle le fait attendre et il n'aime pas du tout cela ! L'attente est réalisée par une suite ininterrompue de combinaisons logiques entre des octets. Lorsque l'interpréteur rencontre une instruction WAIT, il commence par lire un octet sur le port indiqué, puis lui fait subir un OU exclusif (XOR) avec le nombre Y.
Le résultat de cette première opération, tenez vous bien, est combiné logiquement avec le nombre X à l'aide de l'opération ET (AND).
Si le nombre obtenu finalement est 0, alors l'interpréteur recommence tout le cycle, sinon il poursuit son travail avec la prochaine instruction.
Il existe une variante de WAIT où l'on n'écrit pas l'argument Y. Dans ce cas BASIC attend aussi longtemps que (octet lu) AND X soit différent de 0.

2.5. NE CHERCHEZ PAS CES INSTRUCTIONS DANS LE MANUEL !

Ce qui s'est produit pour grand nombre de micro-ordinateurs par le passé, est également arrivé à votre CPC. Un programmeur talentueux a écrit un interpréteur BASIC, est resté pendant de longues semaines devant son écran fignolant avec amour le fruit de ses nuits blanches,

et rage suprême, un auteur de manuel d'utilisation oublie de mentionner certaines instructions.

Cet Oublié s'appelle MOD et vient de "modulo", un terme que les mathématiciens parmi vous connaissent certainement. La fonction modulo fournit le reste d'une division entière.
Voyons quelques exemples:

10 / 4 = 2,5 ou encore 2 reste 2
(en effet: 10 = 2 * 4 + 2)
10 MOD 4 = 2
11 MOD 4 = 3 (11 = 2 * 4 + 3)

Vous pouvez donc utiliser MOD à chaque fois que vous voulez obtenir le reste d'une division entière. Attention, MOD ne fonctionne qu'avec des nombres entiers (de -32768 à +32767).
L'instruction MOD peut par exemple simplifier grandement la programmation de l'algorithme d'Euclide (voir chap. 14).
La contrepartie de MOD est le signe divisé inversé qui réalise la division entière et qui, appliquée par exemple à 11 et 4, donne 2 comme résultat. Ici encore seuls les nombres entiers sont autorisés.

Il a souvent été question des différentes extensions mémoire qui seront disponibles pour votre AMSTRAD et dont les programmes (par exemple des jeux, d'autres langages de programmation etc.) puissent être démarrés à l'aide d'une instruction BASIC. Celle-ci n'est pas mentionné dans le manuel, pourtant elle existe. Contrairement aux autres, cette commande n'est constituée que d'un seul symbole que vous obtenez en pressant simultanément SHIFT et arobas (la touche à droite du "P"). A l'écran vous voyez s'afficher un trait vertical. ce caractère indique à l'ordinateur qu'on veut

s'adresser à un autre espace mémoire. Il nous reste à préciser quelle ROM nous intéresse. Ces dernières possèdent ce que l'on appelle un label ou étiquette qui permet au système d'exploitation
de la distinguer des autres. Puisque dans sa version de base votre CPC ne possède pas d'extension, vous ne pouvez utiliser que le seul label "BASIC". Essayez donc, votre ordinateur réinitialisera l'interpréteur et les lettres "BASIC 1.0" s'afficheront en haut de votre écran. Attention, toutes les données ont été effacées.
Une ROM peut contenir plusieurs labels qui constituent alors en quelque sorte de nouvelles instructions. C'est ainsi que le contrôleur de disquettes (voir chap. 10.1.) possède lui aussi de nouvelles commandes.

Le BASIC du CPC est d'ailleurs étonnemment coopératif. En effet il existe une fonction qui renseigne sur l'adresse à laquelle une variable se trouve en mémoire. Cela ne représente que peu d'avantages pour le programmeur BASIC, mais peut éventuellement devenir intéressant si vous vous tournez vers le langage machine.
Cette fonction est activée par l'intermédiaire de l'arobas: PRINT&A donne l'adresse où se trouve la variable A, dans la mesure où elle a été définie auparavant (sinon apparaît le message INPROPER ARGUMENT).

Voyons maintenant une curiosité: DEC$ (x,y). Cette instruction existe vraiment puisqu'on la retrouve lorsque l'on liste le tableau des instructions qui se trouve en ROM, pourtant il n'existe apparemment pas de routine correspondante dans l'interpréteur. Toute tentative d'appel se solde par un lamentable "SYNTAX ERROR". Peut-être le programmeur avait-il l'intention d'implanter cette instruction en complément à BIN$ et HEX$, puis a opté pour la (meilleure) solution utilisant les préfixes & et &X.

3. LA MEMOIRE

En ce qui concerne la capacité mémoire, le CPC 464 fait partie des ordinateurs les mieux équipés. A côté des 64K de RAM, 32K de ROM se tiennent à votre disposition. Et comble de bonheur, la capacité mémoire est extensible dans des proportions qui laissent rêveur. Comment cela fonctionne et ce que l'on peut en tirer sera le sujet de ce chapitre.

3.1. PROTEGER LA MEMOIRE

Nous avons déjà vu plus haut que 42,5Koctets étaient réservés pour nos programmes BASIC. Mais savoir lesquels de ces octets sont occupés est une opération bien plus délicate. L'interpréteur n'en fait qu'à sa tête pour placer les différents éléments d'un programme BASIC dans sa mémoire. Ne croyez pas qu'il empile sagement ses données dans l'ordre croissant des adresses. Les lignes BASIC se trouvent au début de la mémoire BASIC alors que les "strings" (chaînes de caractères) sont relégués à l'extrémité opposée de la mémoire BASIC. Les autres variables sont logées entre ces deux espaces.
Il arrive un moment où les deux bouts se rejoignent; Le CPC vous signale alors une MEMORY-FULL-ERROR. En fin de compte on ne peut jamais savoir avec certitude quels octets sont libres, et le restent pendant l'exécution d'un programme BASIC.

Si vous désirez inclure un programme en langage machine, il faut par conséquent réserver un certain nombres d'octets qui ne puissent pas

être recouverts. A cet effet il existe la commande MEMORY d'un fonctionnement très aisé. Le sommet de la mémoire BASIC est marqué par un pointeur dont l'interpréteur tient compte lorsqu'il répartit ses données en mémoire. La commande MEMORY permet simplement de modifier le pointeur à notre guise. Si vous rabaissez ce vecteur, qui, soit dit en passant, se trouve aux adresses &AE7B et &AE7C, la mémoire BASIC sera remplie plus tôt.
Connaîssant les adresses, vous pouvez aussi le modifier à l'aide de POKE, au cas où vous seriez adepte de la devise "pourquoi faire simple quand on peut faire compliqué ?".

L'instruction PRINT HIMEM effectue l'opération inverse. Elle nous renseigne sur la valeur actuelle du pointeur. La valeur par défaut est 43903. Vous connaîssez maintenant la dernière adresse de la mémoire BASIC, qui débute à l'adresse 368.

La commande MEMORY ne fonctionne que si on lui fournit un paramètre compris entre 368 et 43903, sinon on obtient le message memory-full-error; c'est une sécurité qui vous évite d'effacer par erreur des données nécessaires au système d'exploitation. De même vous ne pouvez pas réduire la zone à moins de place que n'en nécessite le programme actuellement en mémoire. Bien que cela ait peu d'intérêt, vous pouvez contourner cette dernière protection en POKANT l'adresse dans le pointeur.

Supposons que vous désiriez réserver 256 octets pour un petit programme en langage machine. Il vous suffit alors de taper MEMORY 43647 et le tour est joué. L'espace entre 43648 et 43903 inclus vous est maintenant réservé.

*****

EN BREF: Protéger la mémoire

Avec la commande MEMORY il est possible de déplacer la limite supérieure de la mémoire BASIC. Les valeurs extrêmes sont 368 et 43903.
L'instruction HIMEM renseigne sur la valeur courante de la limite supérieure.

*****

3.2. COMMENT FONCTIONNE LE "BANKSWITCHINHG" ?

Comme nous avons pu le voir aux chapitres précédents, certaines adresses mémoire sont occupées doublement par RAM et ROM. Le microprocesseur quant à lui ne peut adresser qu'un seul domaine à la fois. Il doit donc exister un moyen pour sélectionner l'un ou l'autre. Pour ce faire, les adresses présentes sur le BUS d'adresses sont analysées par un circuit logique qui orientera correctement les données.
Ce circuit peut être commandé extérieurement, c'est à dire que le Z-80 peut décider, grâce à des instructions d'entrées-sorties (semblables à INP et OUT), de quelle manière les adresses vont être décodées. Il arrivera que seuls les circuits RAM doivent être adressées, dans d'autres cas c'est aux ROMs que le circuit connectera le BUS d'adresses. Il est en outre possible de sélectionner indépendamment les deux circuits de ROM. On peut par exemple faire travailler ensembles le système d'exploitation et la RAM vidéo. Le fonctionnement du décodeur logique est comparable à celui d'un aiguillage.

Il est possible de choisir entre les différentes mémoires à partir du BASIC; bien souvent cela aura pour conséquence de "planter" l'ordinateur. Essayez tout de même. Avec OUT &7F82,&82 vous mettez en marche les deux ROMs.

Il n'est donc pas recommandé de commuter entres espaces mémoire en BASIC. En effet, dans la plupart des cas l'interpréteur utilise de la RAM à ses propres fins; déconnecter la RAM au moment crucial ne peut que troubler au plus haut point le Z-80 qui ne trouve alors plus ses programmes. Et une fois l'ordinateur bloqué, il ne reste plus qu'à lui faire le plus énergique RESET qui soit: éteindre la machine.

3.3. LIRE LA ROM

Il est souvent utile ou même nécessaire de pouvoir lire des données de la ROM, par exemple le jeu de caractères (notez au passage que celui-ci se trouve aux adresses &3800 jusqu'à &3FFF). mais ce n'est pas possible à partir du BASIC puisque la fonction PEEK ne lit que dans la RAM. Il n'est pas non plus question d'utiliser le BANKSWITCHING, un plantage étant quasi-inévitable. Théoriquement il serait possible d'écrire un programme lisant la ROM, mais le CPC semble allergique à ce genre de procédé ! Je voudrais tout de même vous proposer un tel listing, car...

a) il montre bien le principe,

b) il fonctionne de manière similaire au programme en langage machine que je vous propose ensuite.

Voici le programme BASIC:

```
1 OUT &7f82, &82
2 A=PEEK(X): REM x=adresse que vous voulez lire
```

Et maintenant un programme qui charge et lance une routine machine ayant la même fonction:

```
1 DATA &01,&82,&7F,&ED,&49
2 DATA &1A,&32,&7F,&AB,&C9
3 MEMORY &AB6F: FOR I=&AB70 TO &AB79
4 READ A: POKE I,A: NEXT: END
5 REM x=adresse que vous voulez lire
6 CALL &AB70,X
7 A=PEEK(&AB7F):RETURN
```

Les lignes 1 jusqu'à 4 initialisent la routine, c.à.d. qu'on abaisse un peu la mémoire BASIC et qu'on écrit, à l'aide de POKE, les instructions machine en mémoire. Cette opération est accomplie une seule fois au début. Ce sont les valeurs derrière les DATAs qui représentent ces instructions (10 octets).

Si maintenant vous voulez lire un octet de la ROM, il faut communiquer l'adresse au programme. Il suffit d'affecter cette valeur à la variable x puis d'appeler la deuxième partie du programme avec GOSUB 6.

En ligne 6 on saute au programme machine. Comme il n'est pas facile de transmettre des données d'une routine machine au BASIC, le programme l.m. dépose son résultat dans une case mémoire de la RAM d'où on pourra la lire avec PEEK(&AB7F), cf. ligne 7.

3.4. EXTENSIONS MEMOIRE

Il existe de nombreuses extensions mémoire pour le CPC sous la forme de ROM (par exemple un programme de traitement de texte) ou encore de RAM permettant d'augmenter la taille de la mémoire utilisateur.

Ce qui est important, c'est que ces extensions sont reliées au BUS d'adresses, et donc adressables par BANKSWITCHING. Le Z-80 n'y verra que du feu, il faut juste lui transmettre les bonnes valeurs avec l'instruction OUT. Le lecteur de disquettes, entre autres, utilise une telle extension ROM contenant les nouvelles commandes.

4. TRUCS POUR L'ECRAN

C'est bien connu, le CPC brille par ses possibilités graphiques. Les instructions existantes autorisent déjà des applications plus qu'honorables, pourtant avec quelques trucs exposés dans ce chapitre on peut aller bien plus loin.

4.1. CARACTERES DE CONTROLE

Avez vous déjà prêté attention au tableau des caractères de contrôle qui se trouve au chapitre 9 de votre guide de l'utilisateur ? Il est possible de se servir de chacun des ces symboles comme d'une

instruction supplémentaire pour la gestion de l'écran. On peut en effet appeler directement certains sous-programmes du système d'exploitation à l'aide de ces caractères spéciaux. L'interpréteur BASIC lui-même se contente de transmettre le code du caractère au système d'exploitation.

Je ne vous présenterai ici que les caractères apportant quelque chose de nouveau; à quoi vous servirait en effet une nouvelle version, moins performante, de LOCATE ?
Tous ces caractères de contrôle ont en commun la manière dont on les appelle: PRINT CHR$(X). Le PRINT fait en sorte que les caractères atterrissent à la bonne routine du système d'exploitation, tandis que le CHR$ sert simplement à convertir le caractère en un nombre (son code).

Le premier caractère (SOH) que nous allons examiner permet de rendre visibles les autres caractères
de contrôle.
Vous savez qu'en principe les caractères ayant un code ASCII inférieur à 32 ne sont pas affichés à l'écran mais transmis comme caractères de contrôle au système d'exploitation. Si maintenant vous faites précéder le caractère par CHR$(1), un symbole graphique va s'afficher; LF par exemple (Line Feed ou avance d'une ligne) est représenté par une flèche vers le bas. Essayez pour voir:

```
PRINT CHR$(1)CHR$(10)
```

L'action de SOH se limite au code lui succédant immédiatement. A chaque fois que vous voulez rendre visible un caractère de contrôle, il faut le faire précéder par SOH.

Le caractère No.2 a aussi un effet très intéressant. Avec STX (c'est son nom) on peut éteindre le curseur. Mais cela ne fonctionne qu'à

l'intérieur d'un programme, car le curseur est rétabli après chaque message "ready". Voici un exemple:

```
10 PRINT CHR$(2)
20 INPUT "TEXTE";A$
30 PRINT CHR$(3)
```

Vous pouvez constater que le curseur est absent lors de l'instruction INPUT de la ligne 20.
La ligne provoque exactement le contraire, c.à.d. qu'elle rend à nouveau visible le curseur (ce qui est absolument sans intérêt dans se programme puisqu'il se termine juste après).

Vous connaissez déjà le caractère No.7 (voir manuel), c'est pourquoi je ne reviendrai pas sur ce joyeux caractère musical.

Intéressons nous maintenant aux caractères provoquant un déplacement du curseur à l'écran.
Voici les équivalences:

```
CHR$(8) = déplacement à gauche
CHR$(9) =      //      à droite
CHR$(10) =     //      vers le bas
CHR$(11) =     //      vers le haut
```

On peut avantageusement utiliser ces codes quand on désire écrire des indices ou des exposants: vous pouvez par exemple monter d'une ligne avec CHR$(11), écrire l'exposant, puis redescendre avec CHR$(10).
A première vue, CHR$(12) peut paraître inutile puisque sa fonction est la même que CLS: effacer l'écran. Pourtant ce caractère
ouvre la voie à d'insoupçonnables applications lorsqu'on l'intègre

dans des variables chaînes de caractères.
Un exemple:

```
A$ = CHR$(12) + "   TITRE"
```

A chaque fois que vous allez afficher (à l'aide de PRINT) la variable A$, l'écran commencera par s'effacer, puis apparaîtra le texte.
D'une manière générale, tous les caractères de contrôle peuvent être intégrés dans des variables du type "string".

Nous passons maintenant à CR (CHR$(13)). CR est l'abréviation de Carriage Return, qui se traduit en bon français par
retour chariot. Le chariot en question nous vient du temps où les ordinateurs servaient essentiellement à commander des télétypes. Notre chariot à nous est le curseur que cette instruction ramène en début de ligne.
Vous connaissez également CHR$(16) qui simule la touche CLR et efface le caractère qui se trouve
sous le curseur.
Essayez vous-même:

```
10 CLS
20 LOCATE 20,10
30 PRINT "espace;espace";
40 WHILE INKEY$="":WEND
50 PRINT CHR$(8)CHR$(8)CHR$(8)CHR$(8)CHR$(8)CHR$(8)CHR$(8)CHR$(16)
```

lancez le programme, appuyez sur n'importe quelle touche et voyez l'effet de la ligne 50.
L'intérêt d'un tel procédé peut paraître douteux, mais il ne tient qu'à vous de lui trouver une application "raisonnable".

Les caractères 17 à 20 effacent des parties d'écran. Remplacez dans le programme précédent le dernier caractère de contrôle par CHR$(17) et lancez le programme. Tous les caractères situés entre le début de la ligne et le curseur (qui se trouve alors sur le ";") sont effacés: le premier mot "espace" disparaît.

On obtient exactement l'inverse en remplaçant CHR$(17) par CHR$(18) qui efface les caractères se trouvant entre le curseur et la fin de la ligne.

Les caractères 19 et 20 ont des fonctions approchantes. CHR$(19) efface tous les caractères entre le début de la fenêtre et la position du curseur, tandis que CHR$(20) supprime les caractères restant dans la fenêtre derrière le curseur. Remplissez l'écran avec n'importe quoi, remplacez le dernier caractère de la ligne 50 par CHR$(19) ou CHR$(20) et regardez le résultat.

Laissons de côté le curseur et passons à CHR$(21), un caractère de contrôle particulièrement attrayant ! En effet PRINT CHR$(21) éteint l'écran texte, ce qui signifie que tout ce qui n'est pas instruction graphique ne sera pas affiché à l'écran.

Voici un exemple d'application pouvant servir de protection à un programme:

```
10 PRINT "Mot de passe:"CHR$(21)
20 INPUT M$
30 IF M$    "top secret" THEN NEW
100 PRINT CHR$(6)
110 ...
```

Après le message de la ligne 10, l'écran texte est arrêté, ayant pour

effet de ne plus afficher les caractères tapés au clavier. Personne ne peut donc lire le mot de passe que vous êtes en train d'écrire. Excepté ce petit détail, l'instruction INPUT fonctionne tout à fait normalement. Le CHR$(6) de la ligne 100 rétablit l'écran texte.

Si vous aimez les effets spéciaux, je vous conseille le signe SYN (code ASCII 22). Si vous le faites suivre de CHR$(1), il établit le mode transparent. Dans ce mode, tout ce que vous écrivez à l'écran se superpose à ce qui s'y trouve déjà. Vous pouvez ainsi souligner des mots à l'écran (une faculté que ne possèdent que de très peu d'ordinateurs). Une fois le mot à souligner écrit, il suffit de faire revenir en arrière le curseur (nous avons vu comment), de passer en mode transparent et de PRINTer le symbole de soulignement (SHIFT 0). PRINT CHR$(22)CHR$(0) supprime le mode transparent.

Encore un caractère de contrôle intéressant: CHR$(24). Il intervertit simplement les couleurs PAPER et PEN. Le résultat est une représentation en vidéo inverse des caractères. Une lettre qui à l'origine était en clair sur fond sombre devient sombre sur fond clair, ce qui correspond plus à nos habitudes d'écriture (ou alors avez vous coutume d'écrire avec de l'encre blanche sur du papier noir ?).
Enfin nous arrivons au petit dernier, qui affecte à nouveau le curseur. PRINT CHR$(30) renvoie le curseur en haut à gauche de l'écran. Cette fonction est également appelée HOME.

Il me reste à vous signaler un petit inconvénient pouvant survenir lors de l'utilisation des caractères de contrôles. Si l'on a employé l'instruction TAG pour envoyer l'instruction PRINT sur le curseur graphique, les caractères de contrôles perdront leur action, ils seront seulement représentés à l'écran par leur symbole graphique.

*****

EN BREF: Caractères de contrôle

CHR$(x)	Fonction
1	Affiche les caractères de ctrl.
2/3	Allume/éteint le curseur
7	Beep
8	Curseur à gauche
9	Curseur à droite
10	Curseur vers le bas
11	Curseur vers le haut
12	Efface l'écran
13	Place le curseur en début de ligne
16	Même fonction que la touche CLR
17	Efface ligne jusqu'au curseur
18	Efface ligne à partir du curseur
19	Efface fenêtre jusqu'au curseur
20	Efface fenêtre à partir du curseur
21/6	Ecran texte éteint/allumé
22	Mode transparent activé/désactivé
24	Vidéo inverse
30	Home

*****

4.2. LA MEMOIRE ECRAN VUE DE L'INTERIEUR

Que savons nous de la mémoire écran ? Pour l'instant pas grand-chose. Ce qui suit est là pour y remédier. Nous avons déjà vu que les points de l'écran étaient représentés par des octets dans la RAM vidéo. La première question qui nous vient à l'esprit: Dans quel ordre les points sont-ils disposés en mémoire ? Pour pouvoir y répondre, faisons une petite expérience. Eteignez puis rallumez votre CPC pour que toutes les données présentes dans la mémoire soient effacées, puis tapez ce qui suit:

```
MODE 2
FOR I = 49152 TO 65535: POKE I,255: NEXT
```

Nous venons de mettre tous les bits de la RAM vidéo à 1, allumant ainsi chaque point de l'écran. Observez l'ordre dans lequel ceci s'effectue. Ce sont d'abord toutes les parties supérieures des lignes de texte qui s'allument, puis on recommence un petit point plus bas, et ainsi de suite jusqu'à ce que toute la surface de l'écran ait été recouverte.

Imaginez maintenant que les 25 lignes de texte soient mises bout à bout pour n'en former qu'une seule. On obtient ainsi 8 lignes constituées de points (car en mode 2 tout caractère est représenté sur une matrice de 8 x 8 points). Si nous divisons les 16K de la RAM vidéo par 8, nous obtenons 2K ou encore 2048 octets pour chacune de ces lignes. Si finalement vous mettez à la queue leu leu ces 8 lignes, vous obtenez l'organisation interne de la mémoire écran de votre CPC préféré. Il nous reste à voir comment un point isolé (pixel) est représenté.

Comme, en mode 2, il suffit de distinguer entre deux couleurs (le point est allumé ou éteint), chaque point est représenté par un bit (0 si éteint, 1 si allumé). Pour une résolution de 640 x 200 points, cela

nous fait 128000 bits ou encore 16000 octets.
Que sont devenus les 384 octets restants ? (16K = 16384 octets). Il y a deux explications à cela. Tout d'abord il est plus simple pour le microprocesseur et pour le circuit TV de travailler avec un nombre entier de Kilo-octets (tout comme nous préférons les nombres entiers aux nombres à virgule), ensuite le Z-80 à besoin de quelques octets en réserve pour certaines opérations comme le scrolling (nous y reviendrons). C'est pourquoi à la fin de chaque bloc de 2K se trouvent 48 octets inutilisés.

En mode 1 nous avons 4 couleurs à notre disposition. C'est la raison pour laquelle on réserve 2 bits pour chaque point, donnant ainsi quatre combinaisons possibles: 00, 01, 10 et 11. Comme nous n'avons que 16K à notre disposition, ce mode est limité à la moitié des points, soit 320 x 200.

Le mode 0 nous permet d'utiliser 16 couleurs. Il nous faut 1/2 octet (un nibble) pour pouvoir les distinguer toutes. Le nombre de bits nécessaires est donc doublé, il ne nous reste alors plus "que" 160 x 200 points.

Le mystère qui entoure la RAM vidéo n'est pas encore tout à fait éclairci. En mode 2 il n'y a pas de problème, chaque octet représente un paquet de 8 points consécutifs. Avec POKE 49152,X on obtient une représentation graphique de la valeur binaire de X. Essayez donc de donner à X les valeurs 1, 2, 4, 8, 16, 32, 64, 128, 240, 15, 170 en notant à chaque fois sur une feuille de papier les valeurs binaires de ces nombres. Vous voyez ?

L'affaire se corse en mode 1. Chaque point nécessite 2 bits. Ne croyez pas que les bits consécutifs aient été regroupés deux par deux pour

former la couleur du point, c'eut été trop simple, et il n'en est rien. Voyons comment il faut procéder pour le décodage. Prenez un octet et divisez le en deux nibbles. Il faut maintenant écrire ces deux nibbles l'un au dessus de l'autre. Cela doit donner ceci (les bits de l'octet ont été numérotés de 0 à 7):

7 6 5 4
3 2 1 0

Les bits superposés donnent la valeur de la couleur. Un octet valant 11110000 représente donc 4 points de la couleur 01 (on lit de bas en haut), le nombre 11111010 aurait donné le premier et le troisième point de couleur 3 (11 en binaire) et le deuxième et le quatrième point de couleur 1. Vous pouvez vérifier tout cela avec un POKE.

En mode 0 le principe est le même. Comme il nous faut 4 bits par points, nous allons diviser un octet en 4 morceaux que nous écrivons les uns au dessous des autres de la manière suivante:

7 6
3 2
5 4
1 0

Veillez à bien respecter l'ordre ci-dessus. Nous avons donc deux points dont la couleur se lit encore une fois de bas en haut. La couleur correspondante vous est donnée dans votre manuel d'utilisation (dans la mesure où les couleurs n'ont pas été redéfinies à l'aide de l'instruction INK).

4.3. DU GRAPHISME CACHE

Lorsque l'on programme du graphisme ou du texte, il arrive souvent que l'on veuille achever complètement la dessin avant de le rendre visible. On encore on désire réaliser deux graphismes complètement indépendants. Il existe deux moyens d'arriver à ses fins. Si les deux dessins ne se recoupent pas, il suffit de maintenir l'un de la même couleur que le fond (avec INK 2,0), et, lorsque tout est prêt, de le rendre visible avec INK 2,24. L'autre dessin subira l'opération inverse.

Malheureusement ce procédé ne fonctionne que si les deux dessins ne se recoupent pas et si le CPC se trouve dans un autre mode que le 2 (en mode 2 il n'est pas possible de définir une couleur de réserve).

Il est plus simple de demander au CPC d'employer une deuxième mémoire écran. Il faut quand même préciser que cette
opération va nous coûter beaucoup de mémoire BASIC. Il est logique que la nouvelle mémoire écran ne puisse se loger qu'en des endroits qui ne sont pas déjà utilisés par l'interpréteur ou par le système d'exploitation. Qui plus est la RAM vidéo ne peut être déplacée que par blocs de 16K, ce qui nous laisse comme adresses de départ possibles: 0, 16384, 32768 ou 49152. Cette dernière est automatiquement sélectionnée à la mise sous tension de l'ordinateur. Les domaines 0 à 16383 et 32768 à 49151 sont déjà occupés, il nous reste donc une seule possibilité: 16384.
Cela implique que nous soyons obligés de réduire la mémoire, avec l'instruction MEMORY 16383, à moins de 16K. La deuxième mémoire écran n'est donc utilisable qu'avec des programmes relativement courts.

On déplace la mémoire écran en 16384 avec l'instruction CALL &BC06,&40.
CALL &BC06,&C0 la ramène à sa position d'origine (49152).

Toutes les instructions graphiques et autres commandes d'affichage concernent toujours la mémoire écran actuellement en fonction, vous pouvez donc travailler tout à fait normalement avec l'une comme avec l'autre RAM vidéo.
Mais on peut on peut aussi travailler dans l'écran désactivé. Il existe à cet effet un vecteur en RAM pointant sur l'adresse de la mémoire écran. Lorsque l'on demande à afficher quelque chose, l'ordinateur ne s'en tient pas à la véritable adresse mais à celle qui se trouve dans le pointeur. On peut ainsi faire croire le système d'exploitation à l'existence d'une RAM vidéo fantôme. Ce fameux pointeur se trouve à l'adresse &B1CB. Tant qu'il contient la même adresse que celle qu'on appelle par l'instruction CALL, tout se passe normalement.
Avec CALL &BC06,&40: POKE &B1CB,&C0 on place la RAM vidéo en 16384, mais l'affichage se fera toujours sur l'ancienne mémoire écran (devenue invisible).
Il est très important de ne POKER que les valeurs &40 ou &C0, car sinon le CPC se plante.

Je veut attirer ici votre attention sur une particularité du CPC. Lorsque l'on "scrolle" l'écran (c.à.d. qu'on fait se déplacer les lignes vers le haut ou vers le bas), l'ordinateur ne décale pas, comme on pourrait le croire, octet par octet le contenu de la RAM vidéo. Le CPC fait plutôt croire au circuit TV que la RAM vidéo débute à une autre adresse, par exemple 53248 au lieu de 49152 pour prendre une valeur arbitraire. Une logique particulière ramène à l'adresse 49152

les octets qui sortent de l'écran. Cette technique pour le moins originale permet un scrolling beaucoup plus rapide. On peut s'apercevoir du décalage d'adresses en POKANT des valeurs dans la mémoire écran. Les points n'apparaissent plus là où l'on pourrait s'y attendre.
A cause de cette technique particulière, il est recommandé, en début de programme, d'envoyer une commande MODE dans chacune des deux mémoires écran. Cela a pour effet de réinitialiser tous les pointeurs; il ne suffit pas d'effacer l'écran avec CLS ou CLG. Par ailleurs, veillez bien dans la suite du programme à ne jamais scroller l'écran.

*****

EN BREF: Déplacer la mémoire écran

CALL &BC06,&40 place la RAM vidéo dans le domaine 16384 à 32767 (ne pas oublier d'abaisser la mémoire BASIC avec MEMORY).
CALL &BC06,&C0 replace la RAM vidéo à son point de départ.
C'est le pointeur se trouvant à l'adresse &B1CB qui indique au système d'exploitation quelle RAM vidéo il doit utiliser.

*****

4.4 SAUVEGARDER LE CONTENU DE L'ECRAN

Il existe dans le BASIC du CPC une instruction permettant de sauvegarder sur cassette le contenu d'une portion de mémoire. Il est donc aussi possible de sauvegarder puis de recharger le contenu de la mémoire écran. Voici les instructions à utiliser:

```
SAVE "!TV",B,49152,16384
LOAD "!TV",49152
```

Vous pouvez bien sûr donner n'importe quel nom au fichier, mais il ne faudra jamais omettre le point d'exclamation, les messages du système d'exploitation viendraient sinon altérer le contenu de l'écran.

Il est également possible de ne sauvegarder que des portions d'écran. Il faut pour cela calculer les adresses de départ et d'arrivée de chaque ligne de points que l'on désire sauvegarder. Avec vos connaissances acquises au chapitre précédent, cela ne devrait pas poser de problème insurmontable. chaque ligne de points est alors sauvegardée séparément avec une commande SAVE et peut être lue à tout moment avec un LOAD. Tout cela ne fonctionne correctement que si l'écran n'a pas été scrollé, vous savez maintenant pourquoi (voir chap. 4.4.)...

4.5. SCROLLING

Tout possesseur de micro-ordinateur connaît le phénomène du "scrolling". Lorsque le curseur arrive en bas de l'écran, tout l'écran se déplace vers le haut pour laisser la place à la ligne suivante. Deux points concernant le scrolling sont intéressant pour le programmeur BASIC. C'est d'une part le principe même du scrolling et son fonctionnement, et c'est par ailleurs la possibilité de scroller dans d'autres direction que vers le haut. Il est ainsi possible d'obtenir des effets très intéressants.

On peut donc également dérouler le texte vers le bas, lorsque l'on tente de faire monter le curseur au-delà de la limite supérieure de l'écran. Dans un programme cela peut être réalisé par des CHR$(11). Avec PRINT on peut ensuite remplir la ligne qui apparaît. Il est aussi possible de déplacer brusquement l'écran en direction verticale. Il suffit d'envoyer deux commandes OUT pour modifier l'offset (=décalage) de l'adresse de départ de la mémoire écran. Comme nous l'avons vu au chapitre précédent, ce ne sont pas les données qui se décalent en mémoire mais plutôt l'adresse de début. C'est ce décalage que l'on nomme offset. Par chance le pas de l'offset peut être inférieur à une ligne. Le plus faible offset possible vaut 2 octets, ce qui correspond, suivant le mode, à un demi, un ou deux caractères. Un scrolling latéral est donc possible.

Le contrôleur vidéo 6845 possède divers registres dans lesquels l'offset est stocké. On peut modifier leur contenu avec OUT, une lecture avec INP n'est pas possible. Voici les instructions:

```
OUT &BC00,13: OUT &BD00, offset
```

La valeur par défaut (c.à.d. si aucun scrolling n'a eu lieu) de l'offset est 0. Si on augmente cette valeur, un scrolling vers la gauche en sera la conséquence, alors qu'une diminution entraînera un scrolling vers la droite. Si vous voulez scroller à gauche et à droite autour d'un point milieu, il est conseillé de commencer par effacer l'écran et de scroller vers le milieu:

```
CLS: OUT &BC00,13: OUT &BD00,20
```

Pour avancer d'une ligne complète, tapez ceci:

```
OUT &BC00,13: FOR I=0 TO 40: OUT &BD00,I: NEXT
```

Le premier OUT sert juste à adresser le bon registre du 6845. Il n'est pas utile de le répéter à l'intérieur de la boucle. Par contre si on y revient plus tard, il faudra à nouveau entrer OUT &BC00,13, car le système d'exploitation aura certainement farfouillé entre temps dans les registres du 6845.

*****

EN BREF: Scrolling

On obtient un scrolling vers le bas en écrivant un CHR$(11) (curseur vers le haut) en haut de l'écran.
Pour un scrolling latéral, on utilise la suite d'instructions

```
OUT &BC00,13: OUT &BD00,offset
```

Offset représente le nombre d'octets divisé par 2 dont on veut décaler l'écran

*****

4.6. "SCROLLING" AUTREMENT

Je vais maintenant vous décrire une propriété du CPC qui devrait faire pâlir de jalousie tous les possesseur d'autres micro-ordinateurs. Elle constitue en quelque sorte le clou en matière de scrolling. Il est en effet possible, tenez vous bien, de déplacer la totalité de l'écran (y compris la bordure !!!) dans les 4 directions (à cet endroit, je

voudrais vous laisser le temps d'exprimer votre jubilation), les constructeurs ont fait du bon travail.

Ici encore les registres du 6845 jouent un rôle capital. Par contre ce n'est plus la mémoire écran qui est modifiée, mais le signal vidéo lui-même. Parmi toutes les informations codées qui constituent ce signal vidéo, se trouvent également des signaux de synchronisation, marquant les fins de ligne et les fins d'image. Si ce signal est retardé de façon incontrôlée, le moniteur ne pourra plus placer correctement les lignes. Résultat: une image déformée.

Le 6845 offre la possibilité de déterminer l'instant des tops de synchronisation au sein du signal vidéo; OUT &BC00,2 est la première instruction à exécuter avant un décalage horizontal. Une deuxième commande (OUT &BD00,x) fixe la valeur du retard, donc du décalage. En utilisation normale x a la valeur 46, mais vous pouvez lui donner n'importe quelle valeur entre 0 et 63. Des nombres plus grands ne lui plaisent pas du tout, et il exprime son mécontentement par un "plantage" de toute beauté. Il ne vous reste plus alors qu'à éteindre ce cher CPC...
Un nombre plus petit que 46 décale à droite, un nombre plus grand décale à gauche. La procédure complète pour glisser d'une unité vers la gauche est la suivante:

```
OUT &BC00,2: OUT &BD00,47
```

Un décalage verticale s'obtient de façon tout à fait similaire:

```
OUT &BC00,7: OUT &BD00,x
```

X doit être compris entre 0 et 38, la valeur d'origine étant 30. Une valeur inférieure fait décaler l'écran vers le bas.

En combinant le scrolling horizontal et vertical, on peut obtenir d'intéressants effets optiques, notamment quand on programme des jeux.

*****

EN BREF: Décaler l'image

Décalage horizontal:

```
OUT &BC00,2: OUT &BD00,x: REM x dans (0-63)
```

Décalage vertical:

```
OUT &BC00,7: OUT &BD00,y: REM y dans (0-38)
```

*****

4.7. REVENONS AU CURSEUR

L'instruction INKEY$ à pour inconvénient par rapport à INPUT qu'elle ne fait pas apparaître le curseur. Heureusement il existe un moyen, même à partir de BASIC, d'allumer ou d'éteindre le curseur en n'importe qu'elle position de l'écran. Deux routines du système ont été prévues à cet effet et il suffit de les appeler. Voici un exemple qui illustre ce que nous venons de voir:

```
10 CAL &BB81: REM allume le curseur
20 WHILE INKEY$ = "": WEND:REM attend un caractère
```

```
30 CALL &BB84: REM éteindre curseur
40 ... suite du programme
```

Tout cela ne fonctionne qu'en mode programmé, car en mode direct le curseur réapparaît après chaque message "ready".

*****

EN BREF: allumer/éteindre curseur

Curseur allumé: CALL &BB81
Curseur éteint: CALL &BB84

*****

5. LE GRAPHISME

Si vous désirez dessiner autre chose que des lignes ou des points, le CPC ne vous aidera pas beaucoup. Curieusement, l'interpréteur BASIC ne connaît pas d'instruction traçant des cercles, des rectangles etc. Heureusement il est très facile de combler ces lacunes par de petits sous-programmes.

5.1. LE CARACTERE DE CONTROLE GRAPHIQUE

Au chapitre 4.1. j'ai volontairement omis de vous parler d'un caractère de contrôle ayant un rapport avec le graphisme haute résolution. CHR$(23) met en marche les différents modes d'encre graphique. Normalement (c.à.d. si CHR$(23) n'a pas encore été utilisé) les points sont simplement affichés à l'écran sans tenir compte de ce qui s'y trouvait auparavant. Mais le CPC peut en faire plus. Si, avant d'allumer un point à l'écran, vous faites précéder l'instruction par PRINT CHR$(23) CHR$(1), alors le nouveau point sera combiné OU-EXCLUSIVement avec l'ancien. Le OU-EXCLUSIF a la propriété intéressante de ne donner pour résultat 1 que si les bits à comparer sont dans un état différent. Si vous aviez à l'écran un ligne entre les points (0,0) et (100,100), le fait de redessiner la même chose en mode XOR effacera la ligne, puisque pour chaque point on aura la couleur 1 XOR 1 = 0 (donc point éteint). Voici à nouveau un programme d'application:

```
10 MODE 2: PRINT CHR$(23)CHR$(1): REM mode XOR
20 FOR I = 100 TO 200 STEP 2
30 MOVE 10,I: DRAW 100,I: NEXT: REM dessine une boîte
40 FOR J = 1 TO 2
50 WHILE INKEY$ = "": WEND
60 FOR I = 130 TO 180 STEP 2
70 MOVE 60,I: DRAW 150,I: NEXT I,J: REM deuxième boîte
```

Vous vous êtes peut-être étonné du STEP 2. Cette instruction est importante, car dans le sens de la verticale on ne peut représenter "que" 200 points, alors que le système de coordonnées est numéroté de 0 à 399 (DRAW 100,100 et DRAW 100,101 produisent la même chose). La suite du programme ne présente pas de difficulté. Au total 3 boîtes sont dessinées à l'écran, la troisième effaçant les deux premières. Voyez vous-même son déroulement.

Il existe deux autres modes, les modes AND et OR. Le principe reste le même, c'est juste la fonction logique qui change. En mode AND, un point ne sera dessiné que si il s'y trouvait déjà auparavant. On peut employer cette méthode en mode multi-couleurs, permettant ainsi de changer de couleur; un point d'une nouvelle couleur n'apparaîtra qu'aux endroits où se trouvaient des points dont la couleur est différente de 0. On sélectionne ce mode avec PRINT CHR$(23) CHR$(2).

Le mode OR (PRINT CHR$(23) CHR$(3)) correspond au mode transparent pour les textes. Cela veut dire que même si le nouveau point a la couleur 0, il n'effacera pas l'ancien !

*****

EN BREF: Modes graphiques

Normal: CHR$(23)CHR$(0)
XOR   : CHR$(23)CHR$(1)
AND   : CHR$(23)CHR$(2)
OR   : CHR$(23)CHR$(3)

*****

5.2. BOITE ET RECTANGLE

Dans ce qui suit je vais vous proposer une série de sous-programmes que vous pourrez utiliser comme de nouvelles instructions. Commençons

par les figures les plus simples: Boîte et rectangle.

Pour nous, une boîte sera simplement une figure à quatre côtés perpendiculaires. Il nous suffit donc de dessiner ces 4 côtés à l'aide de l'instruction DRAW. Voici le sous-programme:

```
65500 REM boite: x1,y1,x2,y2,c
65501 MOVE x1,y1: DRAW x1,y2,c
65502 DRAW x2,y2,c: DRAW x2,y1,c
65503 DRAW x1,y1,c: RETURN
```

Ce sous-programme, ainsi que tous ceux qui vont suivre, est construit de telle manière qu'il puisse être accolé à votre propre programme avec l'instruction MERGE. Celui-ci se chargera également de transmettre les paramètres au sous-programme.

L'aspect de la boîte est déterminé par les coordonnées x1,y1 du coin supérieur gauche et x2,y2 du coin inférieur droit. Il faut également donner à la variable c la couleur de l'encre.

La première ligne de notre sous-programme positionne le curseur au point de départ de notre boîte et tire le premier trait, les trois autres DRAW achèvent le dessin. Enfin avec RETURN on revient au programme principal.
Voici à quoi pourrait ressembler un appel de la routine:

```
x1=100: y1=200: x2=200: y2=100: c=1: GOSUB 65500
```

Le prochain sous-programme dessine un rectangle qui est (par convention) une boîte pleine. La méthode sera de dessiner une

succession de lignes les unes au dessous des autres:

```
65505 REM rectangle: x1,y1,x2,y2,c
65506 FOR ii=y1 TO y2 STEP 2*SGN(y2-y1)
65507 MOVE x1,ii: DRAW x2,ii,c
65508 NEXT ii: RETURN
```

Encore une fois il suffit de donner deux points, tout comme pour la boîte. On retrouve également le "STEP 2" très important en mode XOR dans la boucle FOR-NEXT.
En outre il faut multiplier le pas par -1 lorsque y1 est plus grand que y2, ceci est réalisé par SGN(y2-y1).
En ligne 65507 on commence par placer le curseur graphique sur le côté gauche du rectangle, puis on trace une ligne vers la droite. A chaque appel de la boucle, le curseur descend d'un point de manière à remplir progressivement la surface du rectangle.

5.3. VARIATIONS POUR SINUS ET COSINUS

Les deux prochaines routines vous ont déjà été présentées sous une forme simplifiée dans votre guide de l'utilisateur du CPC. Voici de quoi rompre la monotonie des cercles et autres disques.

Attardons nous encore un peu sur les tracés de cercles. Pour les deux sous-programmes qui vont suivre, il faut fixer les coordonnées du centre (x1,y1), la longueur du rayon (r1) ainsi que la couleur (c). Après l'appel de la routine, la boucle FOR-NEXT parcourt la totalité du cercle (0-360 degrés) et calcule pour chaque point sa

distance au centre (grâce aux fonctions sinus et cosinus). On obtient un disque en remplaçant l'instruction PLOTR par DRAWR, qui tracera alors tous les rayons.

Si l'on choisi un rayon très grand, il peut arriver que certains points du disque aient été "oubliés". On peut y remédier en choisissant un pas plus petit, il faut pour cela ajouter un STEP 0.5 (ou moins) dans l'instruction FOR-NEXT.
Voici les deux routines:

```
65510 REM cercle: x1,y1,r1,c
65511 DEG: r2=r1: FOR ii=0 TO 359
65512 xx=COS(ii)*r1: yy=SIN(ii)*r2
65513 MOVE x1,y1: PLOTR xx,yy,c
65514 NEXT ii: RETURN

65515 REM disque: x1,y1,r1,c
65516 DEG: r2=r1: FOR ii=0 TO 359
65517 xx=COS(ii)*r1: yy=SIN(ii)*r2
65518 MOVE x1,y1: DRAWR xx,yy,c
65519 NEXT ii: RETURN
```

On peut par exemple faire les appels suivants:

```
x1=320: y1=200: r1=100: c=1: GOSUB 65510
```

ou encore

```
x1=100: y1=100: r1=30: c=1: GOSUB 65515
```

Saviez vous qu'avec quelques petites modifications il est également possible de dessiner des ellipses ? Vous avez certainement

remarqué les deux rayons r1 et r2 de la ligne 65512. Lorsque r1 et r2 ont des valeurs différentes, on obtient une ellipse. Faites un essai ! r1 représente le rayon horizontal, r2 le rayon vertical.
Pour simplifier les manipulations, veuillez insérer les lignes suivantes:

```
65520 REM ellipse: x1,y1,r1,r2,c
65521 DEG: FOR ii=0 TO 359
65522 GOTO 65512
```

Le "GOTO 65512" provoque un saut vers la routine "cercle" après avoir fixé les valeurs pour les deux rayons. Cela économise des efforts et de la place mémoire.

Avec une petite variante du programme "disque" nous allons pouvoir dessiner des étoiles avec autant de branches que nous le souhaitons. Au fond un disque est aussi une sorte d'étoile ayant autant de branches que de points sur la périphérie (voir figure 2). Il nous suffit donc de réduire le nombre de branches en insérant un STEP dans le programme. Pour obtenir une répartition uniforme, on prend pour STEP la valeur 360 divisé par le nombre de branches.

Nous utilisons à nouveau un petit rajout, cette fois au programme "disque". Pour s'assurer que toutes les branches seront bien dessinées, on a prolongé la boucle FOR-NEXT jusqu'à 360.

En plus du centre, du rayon et de la couleur il faut transmettre à la routine le nombre de branches, par l'intermédiaire de la variable br. Voici à nouveau le listing et un exemple d'appel:

```
65525 REM étoile: x1,y1,br,c
```

```
65526 DEG: r2=r1: FOR ii=0 TO 360 STEP 360/br
65527 GOTO 65517
```

```
x1=100: y1=100: r1=30: br=12: c=1: GOSUB 65525
```

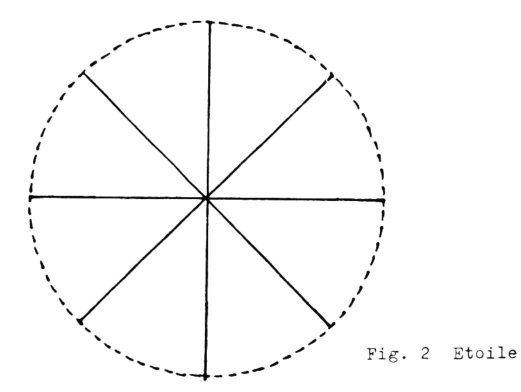


Fig. 2 Etoile

Notre dernier sous-programme dessine des polygones. Si dans le programme précédent nous relions deux à deux toutes les extrémités des branches, nous obtenons un polygone. C'est aussi sur ce principe que fonctionne notre routine. Elle commence par placer le curseur graphique sur le premier point calculé, puis relie chaque point au suivant.

```
65530 REM polygone: x1,y1,r1,br,c
65531 DEG: MOVE x1+r1,y: FOR ii=0 TO 370 STEP 360/br
65532 xx=COS(ii)*r1: yy=SIN(ii)*r1
65533 DRAW x1+xx,y1+yy,c: NEXT ii: RETURN
```

Cette fois il a fallu prolonger la boucle, ici jusqu'à 370.

Si le coeur vous en dit, vous pouvez aussi dessiner des étoiles ou polygones aplatis ou étirés, il suffit pour cela d'introduire un deuxième rayon. On peut également envisager un demi-cercle (ou portion

de cercle). Il faut alors modifier les angles de départ et d'arrivée de la boucle FOR-NEXT, en les remplaçant par exemple par 180 et 360. Vous pouvez laisser libre cours à votre imagination.

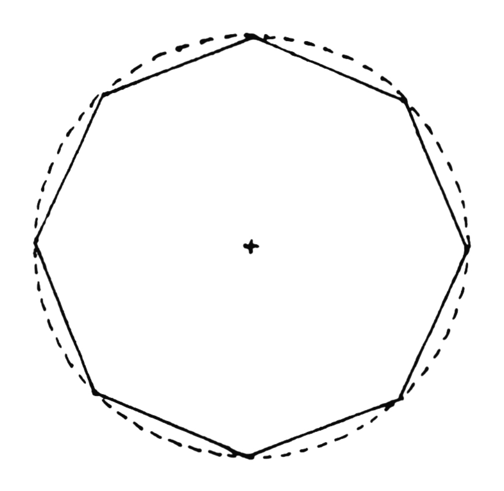


Fig. 3 Polygone

5.4. POURQUOI TESTER UN PIXEL ?

A première vue, l'intérêt des instructions TEST et TESTR ne paraît pas évident. Pour connaître la couleur d'un point ne suffit-il pas de regarder l'écran ? Il existe pourtant des applications irréalisables sans TEST.

Dans ce contexte je pense aux sprites de chez Commodore et aux shapes de chez Apple. Pour ceux qui ne sont pas encore familiarisés avec le monde des micro-ordinateurs, sachez que dans les deux cas il s'agit de figures graphiques que l'utilisateur peut définir puis déplacer à l'écran. Il existe quelque chose de similaire sur votre CPC.
Notre programme doit comporter deux fonctions. Tout d'abord il doit pouvoir recopier n'importe quelle partie de l'écran en un autre lieu. Le principe d'une telle routine est assez simple. Comme au programme "rectangle", on se fixe deux points qui définissent ainsi la zone à recopier. Puis, à l'aide de deux boucles FOR-NEXT, on va lire pixel après pixel tous les points de ce domaine. Le résultat de l'instruction TEST constitue alors la couleur du point à inscrire dans la nouvelle zone.

La deuxième routine doit être capable de faire apparaître à l'écran une figure que ne se trouvait nulle part ailleurs auparavant, elle n'a donc pas été copiée mais créée de toute pièce. Il faut bien sûr que les informations sur cette figure se trouvent notifiées quelque part; en BASIC les lignes de DATAs s'y prêtent très bien. A chaque rangée de points du dessin doit correspondre une chaîne de caractères contenant les informations sur la couleur de ces points. La longueur de la chaîne est égale aux nombres de points en largeur du dessin. Voyons ce

que notre routine doit comporter d'autre. Il faut que la routine puisse détecter la fin des lignes de DATAs; à cet effet nous allons simplement ajouter une ligne vide que le programme saura reconnaître. Il faut également fixer la position du graphisme à l'écran. On transmettra donc les coordonnées par variables.
Voici le premier sous-programme:

```
999 REM recopier zone écran: x1,y1,x2,y2,xs,ys
1000 FOR ii=y1 TO y2 STEP -2: FOR jj=x1 TO x2
1010 PLOT xs+jj-x1,ys+ii-y1,TEST(jj,ii)
1020 NEXT jj,ii: RETURN
```

Et la deuxième routine:

```
1049 REM afficher graphisme: xs,ys
1050 zz=0
1060 READ aa$: ll=LEN(aa$): IF ll=0 THEN RETURN
1070 FOR ii=0 TO ll-1: aa=VAL("&"+MID$(aa$,ii,1))
1080 PLOT xs+ii*m,ys+zz*2,aa: NEXT ii: zz=zz+1: GOTO 1060
```

A l'appel du premier sous-programme il faut transmettre en x1, y1, x2 et y2 les coordonnées du domaine à copier ainsi qu'en zs et ys les coordonnées du point supérieur gauche du nouveau domaine.

Le deuxième sous-programme se contente de moins de paramètres. xs et ys ont les mêmes fonctions que précédemment. Remarquez la multiplication par deux de la variable zz pour éviter que les lignes ne se chevauchent. Quelque chose de similaire se passe pour les coordonnées horizontales. La variable m dépend du mode dans lequel on se trouve et sert de compensation. En mode 0 elle prend la valeur 4, car 4 coordonnées consécutives correspondent au même point sur

l'écran. En mode 1 et 2 m prend respectivement les valeurs 2 et 1. Voici maintenant un exemple d'utilisation du second sous-programme:

```
100 MODE 2
110 xs=320: ys=200: m=1: GOSUB 1050: END
120 DATA "100001001111110100000100000111111"
130 DATA "100001001000000100000100000100001"
140 DATA "100001001000000100000100000100001"
150 DATA "100001001000000100000100000100001"
160 DATA "111111001111000100000100000100001"
170 DATA "110001001100000110000110000110001"
180 DATA "110001001100000110000110000110001"
190 DATA "110001001100000110000110000110001"
200 DATA "110001001111110111110111110111111"
210 DATA ""
```

Vous devez ajouter ces lignes au listing précédent. Dans les DATAs se trouve la trame du mot HELLO que vous reconnaîtrez aisément. Chaque 1 représente un point allumé. Si vous travaillez dans d'autres modes, vous pouvez également rajouter des informations sur la couleur de chaque point.

5.5. SYSTEME DE COORDONNEES

Ce chapitre intéressera surtout les mathématiciens et écoliers parmi vous. Que les premiers me pardonnent de traiter des sujets comme les systèmes de coordonnées de manière aussi peu rigoureuse. Par contre les autres auront peut-être quelques problèmes avec ces notions

nouvelles.

A chaque fois que nous voulons représenter graphiquement une fonction mathématique, il faut d'abord convertir les valeurs en coordonnées pour l'écran. La commande ORIGIN nous y aide quelque peu. Prenons l'exemple de la fonction sinus et voyons ce qu'il faut faire.

Comme vous le savez, le sinus d'un nombre varie entre -1 et 1, c'est pourquoi nous allons placer l'axe des x exactement au centre de l'écran.

En général, pour les fonctions trigonométriques, on place l'axe des y complètement à gauche. L'origine de notre repère va donc se trouver au point (0,199) de l'écran. Nous écrivons par conséquent ORIGIN 0,199 et toutes les instructions graphiques se rapporteront à ce nouveau point zéro.

S'il vous arrive d'oublier les paramètres que vous avez transmis avec ORIGIN, vous pouvez les récupérer avec les lignes suivantes:

```
PRINT UNT(PEEK(&B328)+256*PEEK(&B329))

PRINT UNT(PEEK(&B32A)+256*PEEK(&B32B))
```

Ces lignes fournissent respectivement le X et le Y de ORIGIN.

Si maintenant nous demandions simplement au BASIC de dessiner la fonction sinus (par exemple avec PLOT x,sin(x)), nous n'obtiendrions pas grand chose, l'amplitude maximale de la fonction valant alors 1 pixel. L'instruction serait plutôt:

```
PLOT x,199*SIN(x)
```

Vous pouvez bien entendu multiplier également le x par un certain facteur.

Vous trouverez un sous-programme qui dessine un repère suivant les valeurs que vous lui fournirez.

```
1000 INPUT "X mini";xi
1010 INPUT "X maxi";xs
1020 INPUT "y mini";yi
1030 INPUT "y maxi";ys
1040 CLG
1050 gx=(xs-xi)/639: x=-xi/gx
1060 gy=(ys-yi)/399: y=-yi/gy
1070 ORIGIN x,y
1070 MOVE xi/gx,0: DRAW xs/gx,0: REM axe des Y
1080 MOVE 0,yi/gy: DRAW 0,ys/gy: REM axe des X
1100 RETURN
```

Je ne m'attarderai pas sur les cinq premières lignes dont la fonction devrait être claire.
Les deux lignes calculent les graduations des axes X et Y ainsi que les coordonnées de l'origine (x,y).

Le pas de la graduation se calcule en faisant la différence entre le maximum et le minimum de chaque domaine, divisé par le nombre total de points.
A chaque fois que l'on voudra déterminer la position à l'écran de x ou de f(x), il suffira de diviser par le pas. Il en est de même pour le calcul de la nouvelle origine.

Finalement ligne 1080 trace l'axe des ordonnées (Y) et ligne 1090 l'axe des abscisses (X). On y retrouve également le pas de graduation.

Vous pouvez compléter ce programme en lui faisant écrire la graduation à l'aide d'instructions TAG et PRINT. Les petits tiraits seront dessinés avec PLOTT.

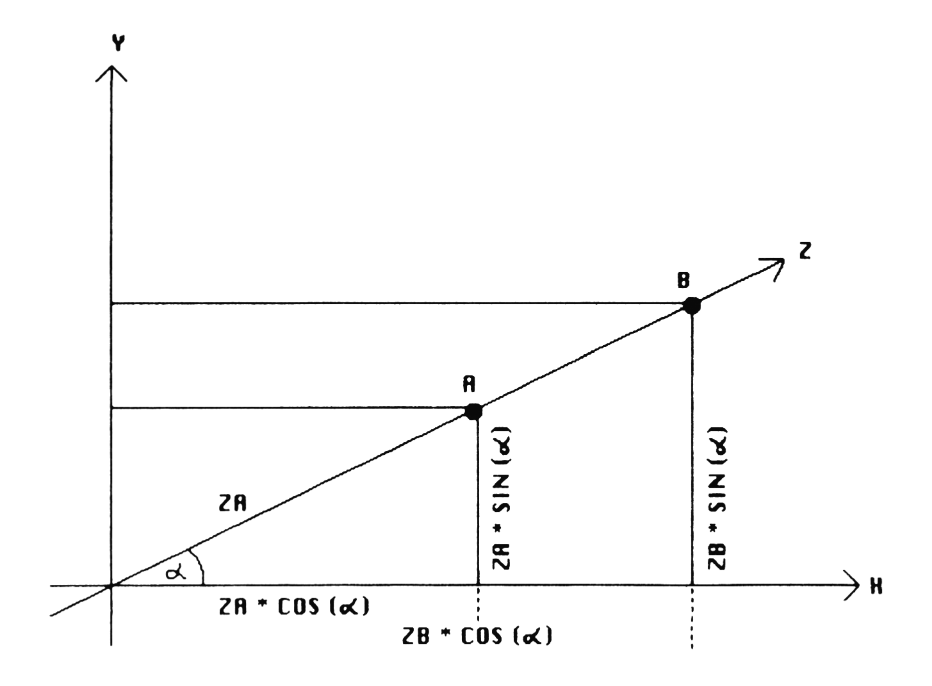


Fig. 4 Repère pour graphismes 3D

5.6. GRAPHISMES EN 3D

La représentation tridimensionnelle des fonctions est sans aucun doute l'application graphique la plus intéressante sur un ordinateur. Malheureusement c'est aussi la plus complexe à mettre en oeuvre. Je voudrais tout de même vous présenter cette technique à l'aide d'un petit programme.

Intéressons nous tout d'abord au système de coordonnées. Contrairement aux fonctions habituelles, nous avons maintenant 3 axes: X,Y,Z.
L'axe Z doit nécessairement être représenté obliquement (voir figure), l'angle (et par conséquent la perspective) reste au choix de l'utilisateur.

Comme il n'est pas possible de transmettre 3 coordonnées à la commande PLOT, nous devons réduire celles-ci à deux dimensions. On appelle cela une projection. Nous nous intéresserons à la projection parallèle, ne comportant pas de point de fuite. Des figures parallèles en réalité le resteront et ne se rejoindront pas à l'infini comme nous le verrions avec nos yeux.

Comme vous le voyez sur la figure, plus la coordonnée Z d'un point sera élevée, plus il sera décalé vers le haut et vers la droite. Vous voyez également que les composantes horizontales et verticales de la projection se calculent de la manière suivante:

x = 0 + zA*COS(a)   y = 0 + zA*SIN(a)

Ou plus généralement:

x = xA + zA*COS(a)   y = yA + zA*SIN(a)

Comme ces ccordonnées 2D ne correspondent pas toujours aux dimensions de l'écran, nous introduisons un coefficient de dilatation (dx,dy dans ligne 100 du listing). L'origine du repère est placée au milieu de l'écran (ORIGIN 320,200), centrant ainsi le dessin.

Nous avons besoin de deux boucles FOR-NEXT imbriquées pour transformer les trois coordonnées d'origine et représenter le tout à l'écran. Pour des raisons que je préciserai plus loin, il recommandé de dessiner la fonction d'arrière en avant.
Dans le programme qui va suivre, toutes les valeurs que l'on modifie souvent sont lues à l'aide d'instructions INPUT. De plus le programme fonctionne en mode 2, car la résolution prime sur la diversité des couleurs. Entrez le programme et démarrez-le. Vous pouvez donner les valeurs suivantes aux variables:

a=45, dx=0.5, dy=0.5, px=2, pz=20

```
10 INPUT "angle";a: DEG: z1=COS(a): z2=SIN(a)
20 INPUT "dilatation-X":dx
30 INPUT "dilatation-Y";dy
40 INPUT "pas-X";px
50 INPUT "pas-Z";pz
60 MODE 2: ORIGIN 320,200
70 FOR z=180 TO -180 STEP -zs
80 FOR x=-180 TO 180 STEP xs
90 y=SIN(x)*COS(z)*100: REM la fonction
100 bx=dx*(x+z*z1): by=dy*(y+z*z2)
110 PLOT bx,by,1
120 NEXT x,z
```

Peut-être avez vous remarqué lors de l'exécution que certains points visibles auraient du être recouverts par d'autres. Ce problème peut être résolu si l'on efface lors de chaque nouveau tracé tous les points se trouvant sous la dernière courbe. Veuillez à cet effet modifier de la manière suivante la ligne 110:

```
110 PLOT bx,by,1: MOVE bx,by-2: DRAW bx,-200,0
```

Un exemple va nous permettre d'y voir plus clair. Supposons que nous ayons une fonction dont la représentation est une surface plane. Cette surface sera donc projetée à l'écran et nous verrons apparaître en perspective un losange incliné vers la droite. Tous les points sont visibles, aucun relief ne vient en masquer une partie. Supposons maintenant que notre surface comporte une bosse. Comme le dessin débutera par l'arrière, il arrivera que des valeurs élevées de la coordonnée Y (représentant "l'altitude") feront que la nouvelle ligne recouvre partiellement la nouvelle, effaçant les points se trouvant "derrière". Le mieux est que vous observiez l'évolution du graphisme pendant l'exécution du programme.

Ligne 10 du programme calcule la valeurs du cosinus et du sinus de l'angle a. Ces valeurs resteront bien sûr constantes par la suite, économisant du temps d'exécution. Vous pouvez modifier à votre guise les nombres en lignes 70 et 80, ils déterminent les bornes du domaine sur lequel vous étudiez la fonction. Veillez tout de même à rester dans les limites de l'écran !
Le facteur 100 de la ligne 90 permet d'étirer ces fonctions qui habituellement restent dans le domaine (-1,1).
Vous pouvez maintenant vous amuser à varier le pas, la dilatation et l'angle de vue. Il est bien entendu également possible de modifier la

fonction elle-même, voici quelques exemples.

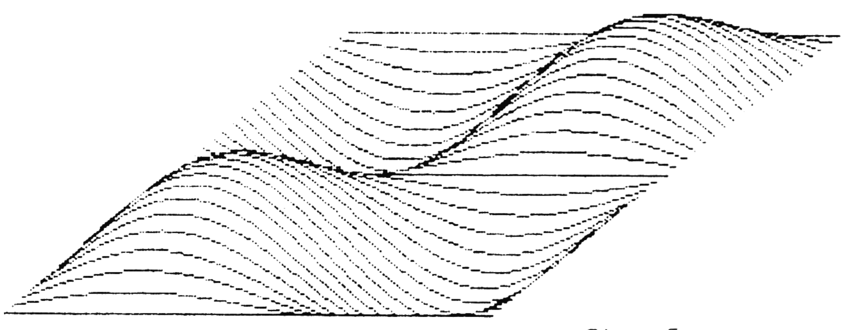


Fig. 5

y = sin(x) * sin(z) * 140

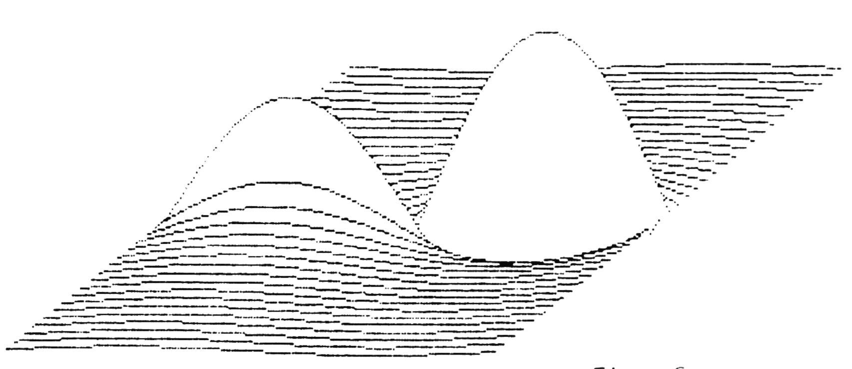


Fig. 6

y = sin(x) * 2 / (z / 8 + 0.5) * 80

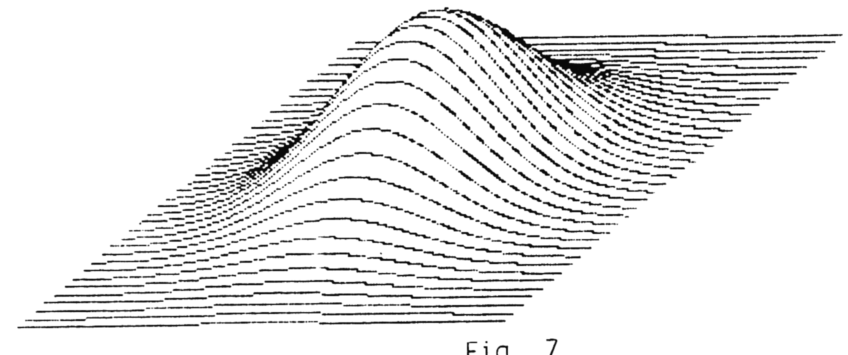


Fig. 7

```
y = exp(-(x*x+z*z)/2)
    /sqr(2*π)*300
```

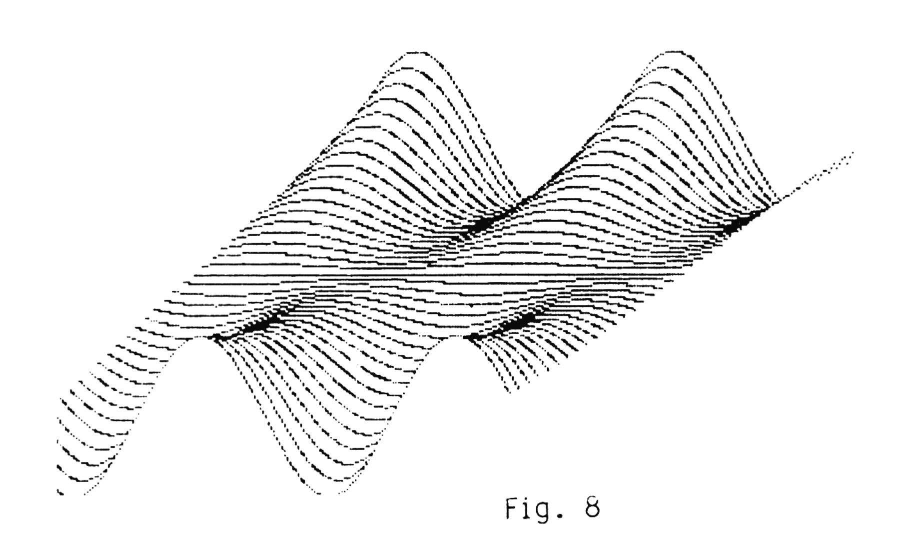


Fig. 8

```
y = sin(x/30)*(exp(z/90)
    -1/exp(z/90))*10
```

6. APPLICATIONS GRAPHIQUES

A l'aide des outils que nous avons vu au chapitre dernier, il a déjà été possible d'obtenir de vraiment beaux graphismes. Toutefois nous n'avons pas encore rencontré d'applications réellement utiles. Si vous faites partie de ceux qui veulent utiliser leur CPC professionnellement, le chapitre 6.1. vous intéressera particulièrement. Les hobbistes et autres artistes n'ont pas non plus été oubliés, ils trouveront au chapitre 6.2. un programme de dessin.

6.1. DIAGRAMMES EN TOUS GENRES

En voyant les courbes et histogrammes qui nous sont proposés à la télévision les soirs d'élections, vous avez peut-être souhaité pouvoir réaliser vous aussi de tels graphismes sur votre ordinateur. Les histogrammes en particulier sont très simples à programmer sur votre CPC. Il suffit en fait de définir chacun des rectangles comme nous l'avons vu au chapitre 5.1. Il nous reste à convertir nos valeurs en coordonnées, ce qui a également déjà été vu au chapitre 5.5.

En abscisse nous n'avons plus besoin d'indiquer un minimum et un maximum, nous aurons seulement à préciser le nombre de rectangles (XS). Ce nombre est ensuite multiplié par 10 pour donner une épaisseur au rectangle (voir ligne 60040).

Les différentes valeurs à représenter sont rangées dans le tableau

V(0...XS-1). La plus grande de ces valeurs est calculée dans la première boucle FOR-NEXT. Etant donné que la numérotation du tableau commence à 0, alors que nous, humbles mortels, commençons toujours à compter à partir de 1, on retranche 1 à XS.

Vous avez certainement remarqué les valeurs 584 et 364 utilisées lors du calcul de l'échelle. Ces valeurs sont volontairement inférieures à 599 et 399 pour laisser de la place à du texte.

En partant du principe que nos résultats ne seront jamais négatifs, nous pouvons donc nous passer du calcul des valeurs minimales (XI et YI).

Ligne 60040 efface l'écran et place l'origine au point (50,30), laissant ainsi de la place à gauche et en dessous du diagramme. Les lignes 60050 et 60055 dessinent la limite supérieure des rectangle, un peu comme nous l'avions fait pour le repère.
Les lignes 60060 et 60070 constituent une variante du programme "rectangle". La différence se situe au niveau du remplissage du rectangle. Etant donné leur configuration, il est en effet plus rapide de tracer des lignes verticales.
J'ai également remplacé quelques variables par un nombre fixe. La valeur 10 représente le nombre de graduations disponibles pour un rectangle. La largeur réelle (en nombre de graduations et non pas en nombre de pixels) est donnée par le "+8" de la ligne 60070. Si vous désirez des espacements de tailles différentes, il suffit de changer cette valeur.

```
60000 REM histogramme: xs,v(0...xs-1),c
60010 xs=xs-1: ys=0
60020 FOR ii=0 TO xs: ys=MAX(ys,v(ii)): NEXT
```

```
60030 gx=(xs+1)*10/584: gy=ys/364
60040 CLG: ORIGIN 50,30
60050 MOVE -5,0: DRAW -5,ys/gy+5,c
60055 MOVE -5,0: DRAW (xs+1)*10/gx+5,0,c
60060 FOR ii=0 TO xs: FOR jj=ii*10/gx TO (ii*10+8)/gx
60070 MOVE jj,0: DRAW jj,v(ii)/gy,c: NEXT jj,ii
60080 RETURN
```

Le deuxième type de diagramme est connu sous le nom de fromage, et il est représenté par des portions plus ou moins grandes de disques. Là encore nous allons pouvoir nous servir de nos anciens sous-programmes.

Le principe en est très simple. A chaque valeur correspond une portion du disque, la somme forme le disque tout entier. Pour pouvoir distinguer entre eux les divers segments, ceux-ci auront chacun une couleur différente.

Vous vous demandez peut-être comment on programme de tels segments de disque, alors que nous n'avons vu jusqu'à présent que le moyen de dessiner un disque en entier. Retournez donc quelques pages en arrière au chapitre 5.3. Dans la ligne 65516 du listing on trouve: FOR i=0 TO 359. Comme je vous l'avais expliqué alors, on parcourt ainsi la totalité du cercle, de 0 à 359 degrés. Si nous changeons la commande en FOR i=180 to 270, nous obtenons un quart de cercle. On peut donc choisir les angles de départ et d'arrivée.

Un petit problème subsiste néanmoins. Les plus observateurs auront certainement remarqué que notre cercle se dessine en partant de "l'Est" et en tournant dans le sens inverse des aiguille d'une montre.

Cela est mathématiquement correct, mais va contre nos habitudes. Nous pouvons contourner ce problème en intervertissant les fonctions sinus et cosinus dans l'expression. Le cercle commencera alors à "12h" et tournera dans le bon sens.
Voici le listing:

```
60100 REM fromage
60101 REM x1,y1,r,xs,v(0..xs-1),c(0..xs-1)
60110 ab=0: DEG: FOR ii=0 TO xs-1
60120 aa=ab: ab=ab+v(ii)*3.6
60125 s(ii)=(ab-aa)/2
60130 FOR jj=aa TO ab
60140 xx=SIN(jj)*r: yy=COS(jj)*r
60150 MOVE x1,y1: DRAWR xx,yy,c(ii)
60155 PLOTR 0,0,1
60160 NEXT jj,ii: RETURN
```

Voyons maintenant les différents paramètres à fournir au sous-programme (la liste se trouve en ligne 60101). x1 et y1 représentent les coordonnées du centre du disque, r le rayon. xs garde sa fonction. Les différentes valeurs à transformer en segment de disque se retrouvent dans le tableau v(0..xs-1) sous forme de pourcentage. Si vous voulez qu'un segment couvre par exemple 20% du disque, il faudra ranger le nombre 20 dans le tableau. Chaque segment peut en outre obtenir sa propre couleur (c(ii)).

Ici encore nous nous retrouvons en présence de deux boucles FOR-NEXT imbriquées, dont celle qui se trouve à "l'extérieur" veille à ce que toutes les valeurs soient prises en compte; l'autre décrit tous les angles des segments.

Les angles de départ et d'arrivée de chaque segment sont calculés en ligne 60120. De plus nous nous permettons un petit luxe. Dans le tableau s(0..xs-1) (que vous devez DIMensionner avant l'appel de la routine) se trouve pour chaque segment l'angle de sa médiane (à mi-chemin entre angle de départ et angle d'arrivée). Ce petit extra vous permettra par la suite d'inscrire, au moyen de TAG, des explication autour de votre fromage ! Il faudra bien sûr veiller à augmenter le rayon pour éviter que les inscriptions ne viennent empiéter sur le fromage.

La seule nouveauté est en fait l'instruction PLOTR. Elle dessine le pourtour du disque au cas où il y aurait un segment de couleur noire. Vous connaissez la suite du programme, je n'y reviendrai donc pas. Veillez tout de même à ce que la somme des valeurs se trouvant dans le tableau v fasse bien 100, sinon le résultat serait faussé.

6.2. UN PROGRAMME POUR "ARTISTES"

Ce programme est destiné à ceux qui veulent exercer leur talent de dessinateur à l'écran. Les graphismes ainsi créés pourront être intégrés dans vos propres programmes.
Il est bien trop épuisant de développer des dessins rien qu'avec les instructions PLOT et DRAW. Ne serait-il pas plus agréable de pouvoir se promener sur l'écran avec les flèches curseur, et d'éteindre ou allumer à volonté des points ? Notre programme sait faire cela.

```
10 MEMORY 16383: CALL &BC06,&40: MODE 2: x=320: y=200:m=1
20 a$=""l WHILE a$="": a$=INKEY$: WEND: PLOT x,y,c
```

```
30 IF a$=CHR$(240) AND y 398 THEN y=y+2
40 IF a$=CHR$(243) AND x 639 THEN x=x+1
50 IF a$=CHR$(241) AND y 1 THEN y=y-2
60 IF a$=CHR$(242) AND x 0 THEN x=x-1
70 IF a$=CHR$(224) THEN c=1: BORDER 24: m=0
80 IF a$=CHR$(13) THEN c=0: BORDER 24,0: m=0
90 IF a$=" " THEN m=1: BORDER 0
100 IF a$="s" THEN CALL &BC06,&C0: MODE 2: INPUT "nom du fichier";f$:
SAVE f$,b,16384,16384: CALL &BC06,&40: GOTO 20
110 IF a$="l" THEN CALL &BC06,&C0: MODE 2: INPUT "nom du fichier";f$:
LOAD f$,16384: CALL &BC06,&40: GOTO 20
120 IF a$="c" THEN CLG: GOTO 20
130 IF a$="x" THEN CALL &BC06,&C0: END
140 IF m=l THEN c=TEST(x,y)
150 PLOT x,y,l: GOTO 20
```

Le dessin est effectué dans la deuxième RAM vidéo, à partir de l'adresse 16384. Cela garantit que vos graphismes créés avec beaucoup de soin ne soient pas recouverts par d'éventuels messages de l'ordinateur.

Ce programme comporte trois modes, qui se distinguent par de différentes couleurs de bordure; ainsi vous savez toujours dans quel mode vous vous trouvez. J'appellerai le premier "mode curseur" car c'est là que vous pouvez déplacer le curseur graphique à l'aide des touches flèchées. Les points de l'écran ne sont pas affectés par cette opération. La bordure est alors sombre. Lorsque la bordure est claire vous vous trouvez dans le "mode dessin", et le point situé sous le curseur graphique est allumé. Dans le dernier mode la bordure clignote et vous effacez ce qui se trouve sous le curseur graphique. Chaque mode peut être appelé à tout moment à l'aide d'une touche. La barre

d'espacement enclenche le mode curseur, la touche COPY le mode dessin et la touche ENTER le mode effacement.

Mais ce n'est pas tout. Tout le contenu de l'écran sera sauvegardé sur cassette lorsque vous presserez la touche S. Le programme commence par sélectionner la mémoire écran d'origine, pour que l'image ne soit pas affectée lorsque vous donnerez le nom du fichier, puis il la sauvegarde (ligne 100). On peut bien entendu également charger une image stockée auparavant sur cassette, la procédure est analogue. Il faut pour cela taper la touche L (voir ligne 101). Les deux lignes comportent une commande MODE qui efface l'écran et bloque le scrolling lors du changement de mémoire écran.

La touche C efface l'écran. Et enfin il nous reste la commande X qui termine le programme sans effacer l'image créée. Si vous voulez retourner au programme pour modifier quelque chose, il suffit de taper:

```
CALL &BC06,&40: GOTO 20
```

Voyons maintenant le fonctionnement du programme. La ligne 10 initialise l'écran et fixe les coordonnées de départ. La ligne suivante attend qu'une touche soit enfoncée et place le nouveau pixel (résultat de la précédente boucle). Puis la fonction (choisie par les IF-THEN) correspondant à la touche tapée est exécutée.

La variable c représente la couleur pour l'instruction PLOT de la ligne 20. Si l'on se trouve dans le mode curseur, cette couleur dépend de la couleur qu'avait l'ancien pixel. De cette manière l'image reste inchangée. L'instruction PLOT a été placée derrière le INKEY$ pour garder le curseur graphique le plus longtemps possible à l'écran. Le

curseur proprement dit est généré par le PLOT de la ligne 150.

Evidemment, ce programme ne constitue pas un super-logiciel de dessin, cela dépasserait le cadre de ce livre. Il est plutôt conçu comme un point de départ pour vos propres développements.
Sachez enfin qu'à l'aide de: LOAD "!nom du fichier",49152 vous pouvez charger sur l'écran normal les dessins créés avec notre programme.

7. PROGRAMMER DES INTERRUPTIONS

La programmation des interruptions est une des caractéristiques les plus marquantes du BASIC Amstrad. Malheureusement le manuel reste très discret sur leur fonctionnement. La description des instructions reste également très succincte. Nous allons donc y remédier au plus vite !

7.1. COMMENT FONCTIONNENT LES INTERRUPTIONS EN BASIC ?

Le principe des interruptions BASIC et des interruptions en langage machine est le même. Dans les deux cas elles sont provoquées par une demande d'interruption (fournie en général par
un TIMER), qui interrompt le programme en cours (il lui laisse tout de même le temps de terminer l'instruction qu'il était en train d'exécuter) et effectue un branchement dans un sous-programme spécial. Jusque là tout est clair. En langage machine l'interruption est déclenchée non pas par programme mais physiquement (c'est un circuit qui produit le signal). Les interruptions BASIC au contraire sont commandées par de savants sous-programmes. De plus celles-ci ne peuvent être déclenchées que par un des quatre (0..3) TIMERs, et non par des signaux extérieurs issus par exemple du bus d'expansion.
Lorsque BASIC se trouve en présence d'un tel signal, il commence par regarder s'il ne doit pas d'abord terminer l'exécution d'une instruction en cours. Ainsi il n'est pas possible d'interrompre des instructions coûteuses en temps comme INPUT.
Il se peut également que la routine d'interruption ait été bloquée par la commande DI (Disable Interrupt). Dans les deux derniers cas

l'interpréteur prend note de la demande et il l'exécutera dès que ce sera possible.
Finalement il sélectionne la routine associée au TIMER venant de s'écouler, et la démarre comme un sous-programme ordinaire.

Mais ce ne sont pas là toutes les tâches que le CPC accomplit en liaison avec les interruptions. Lorsque vous faites clignoter les couleurs avec la commande INK, le changement de couleur est commandé par un TIMER spécifique. Il est possible de régler la vitesse au moyen de SPEED INK.
Autre utilisation des interruptions: la lecture du clavier. Toutes les 1/50 de seconde un circuit interface envoie en microprocesseur le code de la touche venant d'être enfoncée.
Vous voyez, il se passe pas mal de choses dans votre CPC !

7.2. LES INSTRUCTIONS

Après toute cette théorie, nous allons maintenant examiner à la loupe chacune des instruction ayant attrait aux interruptions. Voyons tout d'abord celles qui démarrent l'interruption BASIC: AFTER et EVERY.
Au cas où vous ne maîtriseriez pas complètement la langue de Shakespeare, je vous donne la traduction de ces mots. AFTER signifie "après" et EVERY "chaque" ou "tous les...".
Elles ont toutes deux presque le même effet et la même syntaxe:

```
AFTER x,y GOSUB z
EVERY x,y GOSUB z
```

Et elles règlent toutes deux une minuterie, comme sur nos bons vieux réveils; mais contrairement à ces derniers, la durée se calcule en 1/50 de seconde. C'est le paramètre X. Ainsi AFTER 200... déclenchera une interruption au bout de 200/50=4 secondes.
Cela constitue d'ailleurs la seule différence entre AFTER et EVERY. AFTER ne déclenchera qu'une seule interruption, alors que EVERY lance une suite ininterrompue de cycles de décomptage. On peut par exemple interroger le clavier toutes les 10 secondes alors que presque simultanément des calculs peuvent être effectués.

Le deuxième paramètre (Y) détermine lequel des quatre TIMERs on utilise. Les TIMERs sont en quelque sorte imbriqués les uns dans les autres, un TIMER d'un certain numéro peut interrompre les TIMERs de numéros inférieurs, l'inverse n'est pas possible.
Le reste de l'instruction est clair: il indique ce qu'il faut faire lors d'une interruption.

Le rôle des instructions DI et EI sera vite expliqué. DI est l'abréviation de Disable Interrupt et désactive toute interruption. L'interpréteur prendra néanmoins note si une demande à lieu, et l'exécutera dès que cela sera possible, c'est à dire lorsqu'il rencontrera une instruction RETURN ou EI. Cette dernière signifie Enable Interrupt. Elle autorise les interruptions et exécute toutes celles qui étaient bloquées par la commande DI.
DI et EI servent essentiellement lors d'applications graphiques. On peut ainsi éviter que le curseur graphique ne soit modifié par mégarde lors d'une deuxième interruption.

Pour vous permettre de mieux comprendre la notion de priorité et de blocage d'interruptions, nous allons nous intéresser au petit

programme suivant. Il contient deux routines d'interruptions et le programme principal (ligne 30) est constitué d'une boucle sans fin.

```
1 CLS: PRINT t;"  secondes"
2 LOCATE 6,5: PRINT "secondes"
10 EVERY 50,0 GOSUB 100
20 EVERY 50,1 GOSUB 200
30 GOTO 30
100 t=t+1: LOCATE 1,1: PRINT t
101 WHILE INKEY$="": WEND
102 RETURN
200 x=x+1: LOCATE 1,5: PRINT x
201 RETURN
```

Les deux routines incrémentent toutes les secondes un compteur. La routine en ligne 200 à priorité sur l'autre car elle utilise le TIMER 1. La routine de la ligne 100 nécessite un temps d'exécution plus long car elle attend qu'une touche soit enfoncée (ligne 101).
Lorsque vous lancez le programme vous vous apercevez que la première routine est continuellement interrompue par la deuxième (le compteur en haut de l'écran ne se modifie que lorsque vous pressez une touche).
Insérez maintenant l'instruction DI avant le "t=t+1" de la ligne 100. Toute autre interruption sera interdite pendant l'exécution de la première routine.
Relancez le programme. A présent le deuxième compteur attendra également une touche pour s'incrémenter. En effet la deuxième routine doit attendre que les interruptions soient libérées par le RETURN de la première.

La dernière instruction ayant un rapport avec les interruptions se nomme REMAIN. Elle existe sous deux formes différentes:

a) REMAIN (Y)

b) PRINT REMAIN (Y) (ou a=REMAIN (Y) etc.)

Dans tous les cas on désactive le TIMER Y qui ne peut alors plus déclencher d'interruptions.
Lorsque l'on utilise REMAIN sous forme de fonction (version b) il fournit en outre le nombre de 1/50 de seconde qu'il restait à compter.

7.3. QUELQUES SUGGESTIONS

L'exemple classique d'utilisation des interruptions est certainement la pendule qui s'affiche en permanence dans un coin de l'écran. Il suffit d'appeler toutes les secondes (EVERY 50...) un sous-programme qui incrémente l'heure de 1 seconde, et modifie en conséquence les minutes et les heures. Mais il y a un hic. A chaque opération portant sur la cassette (LOAD, SAVE etc.) la pendule retardera, car pendant ce temps les TIMER sont arrêtés.

Mais que pensez vous d'un jeu de réflexion ou le temps de réponse jouerait un rôle important (un peu comme sur la cassette de démonstration) ? On peut imaginer le déroulement suivant:

1) On pose une question au candidat.

2) Plusieurs réponses possibles s'affichent à l'écran, chacune comportant un numéro. Ce numéro peut être lu avec INKEY$.

3) A l'aide d'une instruction AFTER on interrompt (mettons au bout de 10 secondes) l'attente de INKEY$. Le candidat obtient alors une note, positive ou négative suivant la réponse et le temps de réflexion.

Remarquons au passage que le point 2 ne peut être résolu au moyen de l'instruction INPUT, une interruption ne pouvant jamais interrompre l'exécution d'une instruction; cela est donc aussi valable pour INPUT.

Une autre application serait d'échanger périodiquement deux caractères à l'écran, pour créer l'impression d'un mouvement. Le premier caractère serait un PAC-MAN avec la bouche ouvert, le deuxième aurait la bouche fermée. Alternés périodiquement, ces deux caractères donne l'illusion de la mastication de notre bonhomme.
Les interruptions permettent également de faire déplacer un objet à vitesse constante, quel que soit la complexité du programme en cours.

Vous voyez, la liste des applications est longue, à vous maintenant de trouver des idées géniales.

8. LE SON

Beaucoup de spécialistes considèrent le CPC 464 comme une grande réussite. La production de sons est un détail y contribuant.
Parce que je suis d'avis que pour comprendre ce que l'on programme il faut l'entendre, je ne m'étendrai pas sur les instructions. Je vous renvoie donc aux explications du guide de l'utilisateur et me contenterai d'expliquer les applications qui en découlent.

8.1. UN MINI SYNTHETISEUR

Les instructions ENT et ENV offrent au programmeur une palette quasi illimitée de variations de sonorités allant jusqu'à l'imitation de certains instruments. En effet ces deux instructions règlent tous les paramètres en dehors de la hauteur de la note. Pour vous permettre d'explorer les capacités sonores de votre CPC, j'ai construit le programme "mini synthétiseur" avec lequel vous pouvez modifier tous les paramètres sur simple pression sur une touche.

```
10 MODE 2: WINDOW  1,2,46,2,7: REM menu et fenêtre de travail
20 WINDOW  2,1,40,9,23: REM fenêtre "env"
30 WINDOW  3,42,80,9,23: REM fenêtre "ent"
40 MOVE 0,272: DRAW 639,272
50 MOVE 0,32: DRAW 639,32
60 MOVE 320,32: DRAW 320,272
70 MOVE 368,272: DRAW 368,399
```

```
80 LOCATE 49,2: PRINT "MINI SYNTHETISEUR V1.0"
90 LOCATE 49,4: PRINT CHR$(164)"1985 Data Becker Gmbh"
95 LOCATE 55,5: PRINT "Micro Application"
100 LOCATE 49,7: PRINT "auteur: Hans Joachim Liesert"
110 LOCATE 2,25: PRINT "1=nombre de pas        2=amplitude   du
pas        3=durée du pas"
120 LOCATE 2,3,2: PRINT 2,"ENvelope Volume"CHR$(10)
130 PRINT 2,"          1          2          3"CHR$(10)
140 PRINT 2,"    1     0          0          0"CHR$(10)
150 PRINT 2,"    2     0          0          0"CHR$(10)
160 PRINT 2,"    3     0          0          0"CHR$(10)
170 PRINT 2,"    4     0          0          0"CHR$(10)
180 PRINT 2,"    5     0          0          0"
190 LOCATE 3,3,2: PRINT 3,"ENvelope Tone"CHR$(10)
200 PRINT 3,"          1          2          3"CHR$(10)
210 PRINT 3,"    1     0          0          0"CHR$(10)
220 PRINT 3,"    2     0          0          0"CHR$(10)
230 PRINT 3,"    3     0          0          0"CHR$(10)
240 PRINT 3,"    4     0          0          0"CHR$(10)
250 PRINT 3,"    5     0          0          0"
260 DATA "q","2","w","3","e","r","5","t","6","y","7","u","i"
270 DIM t$(12): FOR i=0 TO 12: READ t$(i): NEXT
280 DIM v(5,3),t(5,3)
290 SPEED KEY 255,10: ENV 1: ENT 1
300 LOCATE 1,1: c=1
1000 CLS 1: PRINT 1,"menu principal"CHR$(10)
1010 PRINT 1,"barre d'espace: jouer mélodie"
1020 PRINT 1,"V              : modifier ENV"
1030 PRINT 1,"T              : modifier ENT"
1040 a$="": WHILE a$="": a$=INKEY$: WEND
1050 IF a$=" " THEN 1100
```

```
1060 IF a$="v" THEN 1200
1070 IF a$="t" THEN 1300
1080 GOTO 1040
1100 CLS  1: PRINT  1,"clavier"CHR$(10)
1110 PRINT  1," 2 3   5 6 7"
1120 PRINT  1,"q w e r t y u i"CHR$(10)
1125 PRINT  1,"barre d'espace: menu"
1130 a$="": WHILE a$="": a$=INKEY$: WEND
1140 IF a$=" " THEN 1000
1150 FOR i=0 TO 12: IF a$  t$(i) THEN NEXT i: GOTO 1130
1160 per=ROUND(125000/440/2p(i/12)): REM le "p" représente "puissance"
1165 c=c*2: IF c=8 THEN c=1
1170 SOUND c,per,0,0,1,1
1180 GOTO 1130
1200 CLS  1: PRINT  1,"modifier ENV"CHR$(10)
1210 PRINT  1,"entrer 0 pour terminer"CHR$(10)
1220 INPUT  1,"quel élément (ligne,colonne)";li,co
1230 IF li*co=0 THEN 1000
1240 INPUT  1,"valeur";v(li,co)
1250 LOCATE  2,co*11-1,4+2*li: PRINT  2,v(li,co);"    "
1260 ENV 1, v(1,1), v(1,2), v(1,3), v(2,1),  v(2,2),  v(2,3),  v(3,1),
v(3,2), v(3,3), v(4,1), v(4,2), v(4,3), v(5,1), v(5,2), v(5,3)
1270 GOTO 1200
1300 CLS  1: PRINT  1,"modifier ENT"CHR$(10)
1310 PRINT  1,"entrer 0 pour terminer"CHR$(10)
1320 INPUT  1,"quel élément (ligne,colonne)";li,co
1330 IF li*co=0 THEN 1000
1340 INPUT  1,"valeur";t(li,co)
1350 LOCATE  3,co*11-1,4+2*li: PRINT  3,t(li,co);"   "
1360 ENT -1, t(1,1), t(1,2), t(1,3), t(2,1), t(2,2),  t(2,3),  t(3,1),
t(3,2), t(3,3), t(4,1), t(4,2), t(4,3), t(5,1), t(5,2), t(5,3)
```

```
1370 GOTO 1300
```

Après le lancement du programme, la fenêtre avec le menu principal (en haut à gauche) apparaît. Pressez une touche correspondant à une des lettres proposées. Tant que tous les paramètres sont à 0, il y a peut d'intérêt à jouer une mélodie. Il faut au moins avoir défini l'enveloppe de volume. Pour modifier le son, on indique au programme le paramètre à changer et la nouvelle valeur. On voit alors les nouveaux paramètres s'afficher dans la matrice au bas de l'écran. Toutes ces valeurs sont directement transmises aux commandes ENV et ENT, il vous est ainsi très facile de les reprendre dans vos propres programmes.

Ce programme relativement long nécessite quelques explications:
Les lignes 10 à 270 définissent les fenêtres et les masques. Deux des trois fenêtres servent à représenter les paramètres des instructions ENV et ENT, la fenêtre restante constitue la zone de travail. En ligne 260 on initialise le tableau t$ qui contient alors le nom des touches affectées à chacune des notes de la gamme. Le programme utilisera le numéro de l'élément du tableau pour calculer le hauteur de la note.

En ligne 280 on dimensionne les deux tableaux réservés aux paramètres de ENV et ENT. La ligne 290 arrête la répétition des touches et annule d'éventuelles enveloppes.

Le programme proprement dit commence en ligne 1000. Il affiche tout d'abord le menu, puis attend un caractère (a$), en fonction duquel il lancera un des sous-programmes.

Les lignes 1100 à 1180 constituent la partie "jouer mélodie". Regardez bien la boucle FOR-NEXT de la ligne 1150. On ne passe à la prochaine

valeur de i qu'à la condition où a$ et t$(i) sont différents. Sinon on calcule à partir de i la période (per) de la note à transmettre à la commande SOUND.

La partie modification des paramètres pour l'enveloppe du volume commence à partir de la ligne 1200. Rien de spécial à signaler ici. Grâce à la technique des fenêtres on peut directement lire les données avec un INPUT. La dernière partie (lignes 1300 et suivantes) sert à la modification des paramètres de ENT, ce qui fonctionne de la même manière que précédemment.

8.2. COMMENT "PROJETER" UN SON ?

Comme vous avez pu le lire dans le manuel du CPC, on peut, en choisissant les bonnes enveloppes, imiter les sonorités de différents instruments. Pour arriver à de tels résultats, il faut beaucoup de patience et un minimum de connaissances sur l'enveloppe de son des instruments à imiter. Ainsi le son d'une trompette chute brusquement dès que l'on arrête de jouer, alors qu'une cloche a une attaque très rapide, puis la note s'éteint peu à peu. Le piano à une enveloppe proche de celle de la cloche, mais un timbre différent, paraissant plus complexe. Dans ce qui suit nous allons essayer d'imiter divers instruments en jouant sur les paramètre de l'enveloppe.

La cloche de verre possède un son très pur, sans variation de fréquence. C'est pourquoi la hauteur du son ne sera pas modifiée par les paramètres de la commande ENT. L'amplitude doit sauter

instantanément à sa valeur maximale, puis décliner lentement vers 0. Les deux matrices de paramètres doivent se présenter comme ceci:

1	15	1		1	0	1
1	0	1		1	0	1
1	0	1		1	0	1
12	-1	8		1	0	1
2	-1	20		1	0	1

Ce qui donne la commande ENV suivante:

```
ENV 1, 1,15,1, 1,0,1, 1,0,1, 12,-1,8, 2,-1,20
```

La cloche de métal a une sonorité plus nasillarde. On obtient un effet approchant en demandant à la commande "enveloppe de ton" de faire varier rapidement la fréquence de la note:

```
ENT -1, 1,1,3, 1,-1,3, 1,0,1, 1,1,3, 1,-1,3
```

Malheureusement, le générateur de sons de votre CPC à un inconvénient. Il n'existe pas de moyen de modifier le timbre (par exemple par filtrage du signal). Au contraire de certains autres micro-ordinateurs, le CPC ne peut produire des son étouffés ou sourds, seulement des sons clairs. C'est pourquoi certains instruments ne peuvent être imités (notamment les bois dans la famille des instruments à vents). De même le son typiquement métallique de la trompette aura bien du mal à sortir de votre Amstrad (en tout cas sûrement pas en BASIC).
Je vous présente néanmoins encore quelques exemples d'enveloppes laissant au moins apparaître de fortes ressemblances avec l'original.

Un harmonica a une attaque plus douce et l'atténuation est également assez lente. De plus le son n'est pas très pur et grésille un peu:

ENV 1, 7,2,1, 1,1,1, 1,0,1, 1,0,1, 15,-1,1
ENT -1, 1,0,3, 1,-1,1, 1,0,2, 1,0,1, 1,1,1

Tous les instruments qui produisent leurs notes par vibration d'une corde ont une enveloppe de volume caractéristique. Après une attaque très rapide, le volume redescend un peu, pour ensuite s'atténuer lentement (ENV 1, 1,15,1, 1,-3,2, 1,0,1, 1,0,1, 12,-1,4). A l'aide de diverses enveloppes de ton on peut alors imiter différents instruments à cordes. Voici un son approchant celui du piano:

ENT -1, 1,1,3, 1,-1,3, 1,0,3, 1,1,3, 1,1,3, 1,-1,3

Le son un peu distordu du banjo s'obtient (avouons-le de manière pas franchement convaincante) avec:

ENT -1, 1,2,1, 1,0,2, 1,0,2, 1,-2,1, 1,0,4

Nous venons d'avoir un aperçu des possibilités qui nous sont offertes. Evidemment vous pouvez programmer autre chose que des instruments de musique. Pourquoi ne pas produire des sons complètement fantaisistes ?

*****

EN BREF: Quelques instruments

Cloche de verre:
ENV 1, 1,15,1, 1,0,1, 1,0,1, 12,-1,8, 2,-1,20
Cloche métallique:

```
ENT -1, 1,1,3, 1,-1,3, 1,0,1, 1,1,3, 1,-1,3
```

Harmonica:

```
ENV 1, 7,2,1, 1,1,1, 1,0,1, 1,0,1, 15,-1,1
ENT -1, 1,0,3, 1,-1,1, 1,0,2, 1,0,1, 1,1,1
```

Instrument à cordes:

```
ENV 1, 1,15,1, 1,-3,2, 1,0,1, 1,0,1, 12,-1,4
```

Piano:

```
ENT -1, 1,1,3, 1,-1,3, 1,0,3, 1,1,3, 1,-1,3
```

Banjo:

```
ENT -1, 1,2,1, 1,0,2, 1,0,2, 1,-2,1, 1,0,4
```

*****

9. BASIC ET SYSTEME D'EXPLOITATION

Il peut être très intéressant d'utiliser à nos propres fins des routines de l'interpréteur et du système d'exploitation; nous en avons d'ailleurs déjà vu quelques-unes par le passé.

9.1. COMMENT SONT STOCKEES LES LIGNES BASIC ?

C'est une question que vous êtes peut-être déjà posé. Nous avons vu au chapitre 2 que les instructions telles que PRINT, SIN, SAVE, etc. possèdent toutes un code spécial. Ce code est appelé TOKEN.
Cette technique permet d'économiser énormément de place mémoire (au lieu de cinq octets pour les lettres, PRINT n'en occupe qu'un seul) et de temps d'exécution (l'interpréteur n'a plus à analyser la suite de lettres).

Les noms de variables et les textes sont stockés tels quels sous forme ASCII. Les opérateurs (*,/,+,- mais aussi AND etc. ainsi que les comparateurs "="," " etc.) possèdent également leur code TOKEN.
La représentation des nombres est beaucoup plus complexe, et dépend de leur type (entier,réel) et de leur taille.

On reconnaît les TOKEN par le fait que se sont des octets supérieurs à 127. Les noms de variables et les chaînes de caractères sont définis à l'aide de signes dont la valeur est inférieure à 128. Ainsi l'interpréteur n'a besoin que d'un seul tableau de codes.

Faisons un petit test. Effacez, avec NEW, tout programme se trouvant éventuellement en mémoire, et entrez la ligne suivante (EXACTEMENT comme elle est écrite ci-dessous):

```
100 PRINT "test"
```

Nous allons maintenant regarder comment ce "programme" est logé en mémoire.
Tapez ceci en mode direct:

```
 FOR i=368 TO 383: PRINT PEEK(i),: NEXT
```

Notre programme commence à l'adresse 368, et nous voyons s'afficher les 16 premiers octets:

```
13 0 100 0 191 32 34 116 101 115 116 34 0 0 0 0
```

Les quatre premiers octets représentent deux pointeurs (de 16 bits chacun). Le premier indique la longueur de la ligne (13 octets). Lorsque l'interpréteur cherche une ligne (lors d'un GOTO par exemple) il commence par regarder la première. Si ce n'est pas la bonne, il lui suffit d'additionner la longueur de la ligne sur laquelle il pointe à l'adresse actuelle, et il arrive alors à la ligne suivante, etc.

Le numéro de ligne (100) se trouve dans le deuxième pointeur. Ensuite vient le premier TOKEN (191=PRINT). 32 et 34 sont les codes ASCII pour l'espace et le guillemet. Comme vous l'avez deviné, le 4 octets suivants représentent les lettres du mot "test". Le 34 est le deuxième guillemet et pour finir on trouve 4 "0" indiquant qu'il n'y a pas d'autres lignes dans le programme.

Nous pouvons manipuler un peu cette structure. Lorsque l'interpréteur trouve une ligne numéro 0 au début de sa mémoire, celle-ci ne pourra pas être listée, bien que l'exécution se passe sans problème. Par contre on ne peut pas faire de saut vers cette ligne et il n'est pas non plus possible de l'effacer. Ainsi, si vous désirez empêcher un listage de la première ligne, il vous suffit de POKEr 0 aux adresses 370 et 371. Essayez, vous verrez.
Ce procédé n'est pas utilisable pour des lignes ne se trouvant pas au début du texte BASIC. Il est possible d'abaisser leur numéro à 0, mais on pourra toujours les lister normalement.

Vous avez peut-être rencontré sur d'autres ordinateurs l'instruction RENEW ou OLD. Elle sert à restaurer un programme effacé avec NEW. Cela fonctionne sur certains ordinateurs (par exemple Commodore) car chez ceux-ci la commande NEW ne remplit pas la mémoire de 0, mais se contente de replacer certains pointeurs. Malheureusement notre CPC s'en tient à d'autres règles de jeu, les programmes effacés avec NEW sont perdus à jamais.

*****

EN BREF: Format d'une ligne BASIC

Toutes les instructions et commandes sont codées en TOKENs, les textes et noms de variables en code ASCII. Les 2 premiers octets représentent la longueur de la ligne, les 2 suivants le numéro de ligne. Avec POKE on peut modifier artificiellement ce numéro. Il est ainsi possible d'empêcher la première ligne d'être listée en forçant son numéro de ligne à 0.

*****

9.2. GARBAGE COLLECTION

Avez vous déjà entendu parler d'une collection de déchets (c'est la traduction de Garbage Collection) ? Si oui, c'était sûrement en rapport avec des ordinateurs. Car ce qui s'apparenterait plutôt à nos voiries publiques, est en fait un dispositif des plus utiles. Pour comprendre cela, il nous faut d'abord connaître quelques notions sur le sujet.

Lorsque l'interpréteur travaille avec des variables string (chaînes de caractères), il produit des déchets à n'en plus savoir où les mettre. A chaque fois qu'un string est modifié d'une manière quelconque (ne serait-ce que pour changer une seule lettre), il est réécrit complètement, l'ancienne version restant inchangée. Il arrive un moment où tous les octets de la mémoire sont pleins, principalement de "détritus". Avant de pouvoir s'occuper d'un nouveau string, l'interpréteur doit d'abord faire le ménage. C'est cela qu'on appelle la Garbage Collection. Et parce qu'elle prend beaucoup de temps d'exécution, elle est bannie par grand nombre d'amateurs. Nous allons illustrer ce problème par un petit exemple:

```
DIM a$(8000)
FOR i=0 TO 8000: a$(i)=CHR$(1): NEXT i
```

Avec ces instructions nous venons de remplir copieusement la mémoire de notre CPC (en fait presque jusqu'à ras-bord !). Il est maintenant possible de déclencher une Garbage Collection avec PRINT FRE(""), mais attention, pas avec FRE(0).

L'inconvénient est que l'exécution de cette instruction apparemment simple dure plusieurs minutes ! En effet l'interpréteur doit effacer 8000 chaînes de caractères sur les 8001 existantes (et toutes identiques). Il est bien plus rationnel de n'en garder qu'une seule et de se rappeler qu'il en existe 8000 semblables.

Pour éviter qu'un programme riche en chaînes de caractères ne soit interrompu pendant de longues minutes par une Garbage Collection, je vous conseille de parsemer de FRE("") les parties utilisant les strings de manière particulièrement intensive (par exemple sous la forme F=FRE("")). Au total la Garbage Collection ne se s'effectue pas plus rapidement, mais elle gêne moins le déroulement du programme.

9.3. ATTENTION: ERREUR !

D'après le vieil adage "pas de programme sans erreur", une petite erreur, bien que discrète, c'est également faufilée dans la ROM BASIC du CPC. Celle-ci concerne les REMs, elle est donc particulièrement difficile à détecter dans les listings.

Lorsqu'après un REM survient un des deux caractères de contrôles "flèche à droite" (touche TAB) ou "barre verticale" (SHIFT + arobas), c'en est trop pour l'interpréteur BASIC et les conséquences sont imprévisibles. Avec un peu de chance les dégats se limitent à un saut en ligne 32511. Si cette ligne n'existe pas, le déroulement du programme est interrompu. Il peut également se produire que certaines parties du programme soient effacées, ou encore que certaines lignes soient rendues invisibles, c.à.d. qu'on ne peut plus les atteindre

avec GOTO, GOSUB, RUN ou LIST.

Les effets de ce "bug" sont très variés et dépendent probablement aussi du reste du programme et de l'endroit où se trouve le REM fautif. Il peut également se produire un changement de mode écran. Peut-être existe-t-il d'ailleurs d'autres caractères produisant de tels effets imprévus.

Il est possible qu'une étude approfondie du phénomène laisse entrevoir un moyen de protéger les programmes contre le listage. On peut aussi envisager l'existence de commandes synthétiques, comme chez le HP-41; des instructions qui n'ont pas été prévus par les concepteurs. Toute indication sur ce sujet serait la bienvenue.

9.4. COMMANDE INCONNUE

Tout comme les instructions inconnues du chapitre 2.5., je voudrais vous présenter une particularité du système d'exploitation qui n'est pas mentionnée dans le guide de l'utilisateur.

Elle concerne l'éditeur, responsable entre autres des déplacements du curseur, de la touche COPY et de la gestion des lignes que vous tapez au clavier. Lorsque vous avez déjà tapé quelques caractères d'une nouvelle ligne, ou que vous avez utilisé la commande EDIT pour en modifier une, il vous faut d'abord aller placer le curseur au bon endroit. Tous les caractères que vous tapez maintenant sont insérés, repoussant ceux qui se trouvent derrière le curseur. Cela peut devenir très gênant lorsque l'on désire simplement remplacer les anciens

caractères par les nouveaux. Par chance il est très facile de désactiver cette insertion, il suffit de presser simultanément les touches CTRL et TAB. Une deuxième action réactive l'insertion.

9.5. COMMENT TROMPER LE BASIC

Comme tout autre programme, l'interpréteur BASIC se sert de certaines cases mémoire pour y entreposer des données. En modifiant ces octets avec des POKEs, il est possible de faire faire pas mal de choses à BASIC.

Prenons par exemple la protection de programmes implémentée dans le BASIC du CPC. Lorsque l'on veut charger un programme protégé, l'interpréteur en prend note en modifiant un octet spécial. Arrivé à la fin du programme cet octet est lu, et le cas échéant BASIC envoie un NEW. L'octet en question se trouve à l'adresse &AE45. Si il contient une valeur différente de 0, le programme en mémoire est protégé. Avec POKE &AE45,1 il est donc possible d'activer la protection, POKE &AE45,0 provoque l'inverse.

L'octet avec l'adresse &AC00 possède une propriété très intéressante. Suivant la valeur de son contenu, l'interpréteur supprime les espaces inutiles dans les lignes BASIC ou alors il les conserve (cas normal). POKE &AC00,1 active cette routine très utile et économe.

Au chapitre 3.1. nous nous étions penchés sur le fonctionnement de la

fonction HIMEM qui permet d'abaisser la limite supérieure de la RAM BASIC. Il est également possible de remonter le début de la RAM BASIC. Il n'existe bien sûr pas d'instruction pour cette fonction, mais elle est réalisable et vous ouvrira peut-être de nouvelles possibilités.
Les adresses &AE81 et &AE82 constituent un vecteur pointant sur le début du programme. Normalement il indique l'adresse 367, l'octet précédant le début du programme. En modifiant le pointeur nous ne changeons rien à la mémoire, mais l'interpréteur ira chercher ailleurs le début du programme. Il ignorera complètement toute la partie du programme se trouvant avant la nouvelle adresse de départ. De cette manière on peut cacher des portions de programme, à condition bien entendu de connaître les adresses de fins des lignes à cacher.

Il est très simple de faire disparaître tout le programme. On donne pour cela au pointeur de début de programme la valeur des registres de fin de programme (&AE83 et &AE84):

```
POKE &AE81,PEEK(&AE83): POKE &AE82,PEEK(&AE84): NEW
```

Le NEW sert à effacer les anciennes variables que BASIC risquerait d'interpréter comme lignes de programme. Il n'affecte pas le programme puisque les pointeurs ont été placés avant.
Il est maintenant possible de charger un deuxième programme sans influencer le premier. Seules les variables sont perdues.

```
POKE &AE81,111: POKE &AE82,1
```

rétablit l'état initial.

Les programmes cachés de la sorte ont une propriété intéressante. Lorsque les pointeurs de début de programme sont modifiés pendant l'exécution du programme, celui-ci n'en est presque pas affecté. Seules les instructions de branchement ne fonctionnent plus, car l'interpréteur a besoin des pointeurs pour se repérer.

*****

EN BREF: Jouer un tour à BASIC

&AE45 indique si il y a protection ou non.
POKE &AC00,1 comprime les lignes de programme (est désactivé par POKE &AC00,0).
Les octets &AE81 et &AE82 pointent sur le début du programme, les octets &AE83 et &AE84 sur la fin.

*****

9.6. ENCORE QUELQUES TRUCS

Lorsque l'on "bricole" sur un programme, on oublie bien souvent les valeurs que l'on avait données à certains paramètres. En ce qui concerne l'instruction SPEED INK je peux vous proposer un remède. En effet les deux paramètres de cette instruction sont stockés aux adresses &B1D7 et &B1D8. C'est là que le système d'exploitation vient les lire, à chaque changement de couleur. Et ce qu'un système d'exploitation sait faire, nous savons le faire aussi, à l'aide de PEEK...

Vous avez certainement déjà fait usage de la possibilité de redéfinir des caractères. Vous avez alors sûrement regretté comme moi qu'il faille redéfinir les 8 octets de la matrice même si l'on ne veut changer qu'un seul bit. Et bien, une fois de plus, ce problème peut être contourné. Les caractères redéfinissables CHR$(240) à CHR$(255) se trouvent en RAM aux adresses &AB80 jusqu'à &ABFF. Chaque caractère occupe 8 octets que nous pouvons lire avec PEEK. Il suffit de modifier l'octet contenant le point à changer.

On détermine l'adresse du caractère grâce à la formule suivante:

Adresse = 43904 + (X-240)*8

X est le numéro du caractère à modifier et doit être compris entre 240 et 255.

*****

EN BREF: Trucs pour système d'exploitation

Les paramètres de la commande SPEED INK se trouvent aux adresses &B1D7 et &B1D8.
Les matrices des caractères redéfinissables sont en RAM aux adresses &BA80 à &BAFF.

*****

10. LES PERIPHERIQUES ET LEUR FONCTIONNEMENT

Tout ordinateur a besoin d'un certain environnement (périphériques) pour pouvoir travailler correctement. Chez certains fabriquants, comme par exemple IBM, il faut quasiment monter son ordinateur en kit car même le clavier et le moniteur de l'IBM-PC sont livrés séparément. Amstrad par contre a fait vraiment fort un fournissant un moniteur et un lecteur de cassettes en série. Néanmoins, il est encore possible d'adjoindre des périphériques au CPC.

10.1. LE LECTEUR DE DISQUETTES

Est-ce que vous faites partie de ces personnes pressées qui trouvent que le chargement de programmes par cassettes est beaucoup trop long ? Est-ce que vous ressentez aussi ce tressaillement nerveux lorsque le crépitement du haut-parleur vous signale que le CPC est en train de charger le 26ième bloc ? Alors il faut vous procurer un lecteur de disquettes. Tout amateur averti connaît l'extraordinaire confort qu'un lecteur de disquettes apporte par rapport aux cassettes. Pour vous en convaincre, sachez qu'un programme de 20K, qui se charge sur cassette en 2 minutes 27 secondes avec SPEEDLOAD, se contente de 9 malheureuses secondes avec le lecteur de disquettes.

Mais la rapidité n'est pas le seul avantage. Comme pour un disque musical, tout point de la disquette peut à tout moment être lu, il suffit de placer la tête de lecture sur la bonne piste (il y en a 40), permettant ce que l'on appelle l'accès direct. Cela

signifie qu'on n'a pas besoin de lire toutes les données d'un fichier pour aller en chercher une particulière, il suffit quasiment de dire à l'ordinateur "vas me chercher le 3ième octet du fichier d'adresses". Bien sûr il faut le lui dire dans son patois, mais ceci fait, il exécutera notre ordre en quatrième vitesse. De cette manière on peut presque considérer la disquette comme un extension de la mémoire.

A côté de tous les autres avantages d'un lecteur de disquettes (comme par exemple le catalogue des fichiers, qui se trouve sur chaque disquette), il faut également signaler un petit inconvénient. Un tel lecteur est cher.
Tout lecteur de disquette nécessite un circuit électronique de commande appelé contrôleur. Chez Amstrad le contrôleur est fourni avec le premier lecteur que vous achèterez (le plus cher); il peut gérer simultanément deux lecteurs, c'est pourquoi vous pouvez brancher un deuxième drive sur le premier.
Le boîtier du contrôleur contient également une ROM dans laquelle se trouvent les nouvelles instructions spécifiques aux disquettes.

10.2. L'IMPRIMANTE

La sortie imprimante fait partie des nombreux atouts du CPC. Au contraire du floppy-drive elle ne nécessite ni contrôleur ni nouvelles instructions. Tout est intégré dans la machine.
De plus il s'agit ici d'une interface Centronics,
le standard le plus répandu chez les fabriquants d'imprimantes. Vous pouvez donc vous connecter à la majorité des imprimantes présentes sur le marché.

Mais comme partout, il y a encore un petit hic. Le CPC ne transmet que les 7 premiers bits sur les 8 qui composent un caractère ASCII et vous prive par conséquent de la moitié des caractères représentables. Mais rassurez vous, les codes 0 à 127 contiennent tous les caractères importants, ceux qui manquent sont en général des caractères graphiques.

Il arrive aussi que le code ASCII de l'imprimante ne corresponde pas exactement à celui du CPC. Dans ce cas vous devez établir un tableau de correspondances. Ne vous fiez pas aux apparences, c'est plus simple qu'il n'y paraît. Reportez simplement les codes particuliers à votre imprimante dans un tableau (0..127) de nombres entiers. Les caractères à imprimer seront alors envoyés lettre par lettre avec la commande PRINT 8,CHR$(X). X est le code ASCII de la lettre souhaitée. Pour envoyer un code modifié, remplacez le X par TABLEAU(X).

10.3. LES MANETTES DE JEUX

Comme beaucoup d'autres passionnés de micro-informatique, vous avez peut-être un jour écrit un programme qui interroge le Joystick. Probablement avez vous alors testé les différentes positions avec des "IF-THEN":

```
IF JOY(0)=1 THEN 100
IF JOY(0)=2 THEN 200
IF JOY(0)=4 THEN 300
...
```

Cette méthode est très lente et relativement compliquée. Que pensez vous du programme suivant ?:

```
10 a=LOG(JOY(0))/LOG(2)
20 ON a GOTO 100,200,300,...
```

La ligne 10 réalise le bon branchement en fonction de la variable a. La ligne 10 sert à transformer les valeurs fournies par le joystick (croissant par puissances de 2: 1,2,4,8,16,32) en une suite arithmétique (1,2,3,4,5,6).
Le fonctionnement d'une manette de jeux est très simple. Il est constitué de 5 ou 6 interrupteurs (de qualité d'ailleurs très variable suivant les modèles). Pour chaque position du manche, un des interrupteurs sera actionné et le CPC enregistrera la valeur correspondante.
Les interrupteurs utilisés vont du simple contact à membrane sur les joysticks les moins chers jusqu'aux micro-rupteurs sur les plus performants. On les reconnaît en général au petit "clic" qu'on entend.
Lors de l'achat d'une manette de jeux, veillez à ce qu'elle comporte si possible des angles arrondis, car sinon vous vous fatigueriez très vite.
Tous les joysticks compatibles au standard Atari sont utilisables (se sont de loin les plus répandus).

11. LES PORTS D'ENTREE/SORTIE

Sur la face arrière de votre CPC se trouvent divers connecteurs et prises. Dans ce chapitre nous allons voir comment tout cela

fonctionne.

11.1. TOUR D'HORIZON DES INTERFACES

Avant de nous attaquer aux différents connecteurs, intéressons nous pour commencer à la notion d'interface.

Il s'agit dans tous les cas d'une liaison vers les périphériques, autorisant en général une interaction assez "intime" entre le CPU et le monde extérieur. Le meilleur exemple en est le BUS- ou port d'expansion. Sur le boîtier il porte l'inscription lapidaire "floppy-disc". Mais il est possible de brancher bien d'autres choses que le lecteur de disquettes. C'est là que seront logés les cartouches de ROM supplémentaire, et celles-ci font partie de l'environnement immédiat du microprocesseur.
C'est pourquoi le port d'expansion est "uniquement" constitué des BUS d'adresses, de données et de commande du Z-80, plus quelques autres lignes de commandes.

Juste à côté nous avons la sortie imprimante. Elle correspond au standard Centronics. On retrouve ici le BUS de données, mais pas les deux autres. Ceux-ci sont remplacés par quelques lignes de commande venant du circuit interface 8255.

Malheureusement il a fallu que ce soit justement sur cet élément que les concepteurs du CPC aient fait des économies. Comme nous l'avons déjà vu, seuls les 7 premiers bits de données sont transmis, le

dernier à purement et simplement été "oublié" (mais connaissez vous un ordinateur parfait ?).

La sortie manette de jeux est simplement une extension du clavier. Elle est reliée à divers circuits dans le CPC (nous y reviendrons).

Les sorties vidéo et son sont aussi des sortes d'interfaces, mais elles ne travaillent pas directement avec le microprocesseur.

11.2. COMMENT FONCTIONNE UNE INTERFACE ?

Le principe de fonctionnement d'une interface est le même pour tout ordinateur. Un circuit d'interface transmet vers le périphérique l'information recueillie sur le microprocesseur. De plus il peut s'avérer nécessaire de synchroniser les deux appareils. Dans ce cas ceux-ci échangent (sur des lignes de commande spéciales) des impulsions, indiquant ainsi qu'ils sont prêts à recevoir ou transmettre des données.

Avant chaque opération de transfert, le microprocesseur doit indiquer s'il s'agit d'une émission ou d'une réception. En fonction de cela le port sera configuré en entrée ou en sortie.

Il arrive que pour certaines tâches, le CPC n'ait pas besoin du circuit d'interface; c'est le microprocesseur qui prend alors en charge toutes les opérations; l'exécution en est bien sûr moins rapide. Dans ce cas les éventuels circuits logiques se contentent d'adapter les différents niveaux de tensions.

Tous les programmes nécessaires à l'utilisation des interfaces existent déjà dans la ROM, et peuvent être appelés par des instructions BASIC. Mais une programmation vraiment efficace ne peut s'effectuer qu'en langage machine.

11.3. UNE INTERFACE PERSONNALISEE

Est-ce que vous aussi faites partie de ces incorrigibles spécialistes "hardware", pour qui un ordinateur évoque avant tout fers à souder chauds, circuits imprimés complexes et bricolage un perspective ? Les représentants de cette espèce "d'homo electronicus" ont souvent le désir de charger l'ordinateur avec des données provenant de quelque appareil de mesure, par exemple d'un thermomètre électronique. A cet effet il existe l'interface Centronics et le BUS d'expansion. Mais en pratique, tous deux sont déjà prévus pour d'autres occupations.

Cela est également vrai pour la sortie (qui est en fait une entrée) joystick, mais elle n'est en général utilisée que pour les jeux. Il est alors possible de l'utiliser à des fins sérieuses.

Le user-port (c'est l'inscription que vous trouverez gravez noir sur noir à l'emplacement de la sortie manette de jeu) peut très facilement être lu en BASIC avec les fonctions JOY et INKEY. Toutes les conditions sont donc réunies pour que nous puissions entrer en contact avec le monde extérieur au CPC.

A chacune des broches 1 à 7 on peut relier deux interrupteurs. Les

premiers ont leur point commun reliés à COMMON (broche 8), les seconds utilisent COM 2 (broche 9). C'est à l'aide de ces deux derniers contacts que l'Amstrad fait la différence entre joystick 1 et 2. En actionnant un interrupteur on relie une entrée (1 à 7) à l'un des deux signaux COMMON. Pour le CPC cela représente exactement la même chose que la pression sur une touche du clavier, vous pouvez vous en assurer avec les instructions mentionnées plus haut.

Vous pouvez donc brancher n'importe quels interrupteurs sur les entrées du port joystick. Le choix des appareils actionnant ces contacts (relais, transistors etc.) est laissé à votre libre imagination.

11.4. LE CLAVIER

Pour compléter ce tour d'horizon, je voudrais maintenant vous expliquer comment fonctionne la lecture du clavier.
Faites le compte, le clavier du CPC comporte 73 touches (en ne comptant qu'une seule fois les deux touches SHIFT), qui doivent toutes être lues à intervalles réguliers (1/50 de seconde). Le clavier est décomposé en une matrice de 10 colonnes, que le Z-80 peut activer, il lui suffit de donner au circuit d'interface 8255 le numéro de la colonne.

Lorsqu'une touche de la colonne sélectionnée est enfoncée, le bit correspondant dans la matrice est mis à 0, sinon il reste à 1. Par colonne on trouve ainsi un maximum de 8 bits, formant un octet que le circuit générateur de sons (si,si, vous avez bien lu) renvoie au Z-80

via le 8255. Il est alors possible au microprocesseur de décoder les octets reçus et d'en déduire la touche enfoncée. Ce procédé à première vue tiré par les cheveux, passer par l'intermédiaire du générateur de son, à été choisi pour décharger les autres ports. Celui-ci dispose en effet lui aussi d'un port, relativement lent, mais suffisamment rapide pour l'utilisation qui lui est demandée ici.

Le principe de la lecture clavier par rangées est d'ailleurs utilisé sur la grande majorité des ordinateurs, avec plus ou moins de variantes.

12. LE DATACORDER ET LE CLAVIER

Le maniement du lecteur de cassette du CPC est traité de manière plutôt sommaire dans le manuel. Nous allons y remédier ici. La lecture du clavier possède peut-être aussi encore quelques secrets pour nous que nous allons examiner.

12.1. COMMENT CREER UN FICHIER ?

En feuilletant le guide de l'utilisateur, vous êtes certainement tombé sur des fichiers du type "ASCII". Malheureusement les explications se limitent uniquement à l'application aux traitements de textes, alors que les fichiers ASCII se prêtent très bien à l'enregistrement de variables numériques et de chaînes de caractères. Je voudrais également vous montrer comment charger un fichier ASCII à partir du BASIC.

Le nom "fichier ASCII" vient du fait que toutes les données sont enregistrées comme une suite de caractères ASCII. Contrairement à un fichier programme qui est simplement une "image" d'une portion de mémoire du CPC, chaque écriture est terminée par l'envoi d'un CHR$(13). Peu importe si les données sont des variables ou autre chose.

L'écriture et la lecture de tels fichiers ressemble fort aux commandes écran: on utilise les instructions INPUT 9 et PRINT 9 qui doivent être, comme à l'écran, validées par un CHR$(13) (Carriage Return).

Supposons un peu que dans votre obstination, qui caractérise tout authentique mordu d'informatique, à vouloir collectionner un maximum de données, vous ayez l'intention de sauvegarder pour la postérité deux variables remplies de données. Voici comment il faut procéder:

```
10 REM créer fichier
20 OPENOUT "test"
30 PRINT  9,a$
40 PRINT  9,b
50 CLOSEOUT
```

```
10 REM relire fichier ci-dessus
20 OPENIN "test"
30 INPUT  9,a$
40 INPUT  9,b
50 CLOSEIN
```

Le premier programme enregistre sur cassette les deux variables a$ et b; celles-ci pourront être relues avec le deuxième programme, même sous d'autres noms, par exemple x$ et y. Il importe seulement que le type des variables soit le même que lors de l'écriture, sinon on obtient un TYPE MISMATCH ERROR.

Pour la lecture de fichiers ASCII, on procède de manière analogue, voici encore un exemple de programme:

```
10 INPUT "Combien de lignes";a: a=a-1
20 DIM a$(a)
30 OPENIN "nom du fichier"
40 FOR i=0 TO a
50 INPUT  9,a$(i)
```

```
60 NEXT
70 CLOSEIN
```

Après l'exécution de ce petit programme, les lignes d'un texte ASCII se retrouvent dans le tableau a$. Malheureusement cette façon de faire possède un inconvénient. En effet BASIC se sert de virgules comme caractères de fin de ligne. C'est pourquoi toutes les lignes comportant une virgule (et il y en a certainement pas mal) seront découpées aux emplacements de la virgule, le reste allant dans le string suivant. Rassurez vous, le CPC tient à votre disposition la solution à ce problème. Il suffit de remplacer le INPUT par un LINE INPUT et le tour est joué. L'interpréteur ne reconnaîtra alors plus que le CHR$(13) comme fin de ligne.

Dans le programme ci-dessus il fallait indiquer soi-même combien de lignes le fichier contient. Si vous vous trompez dans ce nombre, il se peut que le CPC vous affiche un "EOF met" dans le cas où le programme tente de lire plus de données qu'il ne s'en trouve dans le fichier. Il existe en BASIC une fonction qui indique la fin d'un fichier: EOF (End Of File). Tant que toutes les données n'ont pas été lues, PRINT EOF donnera 0 comme résultat, après la dernière donnée cela donne -1. Apportons la modification suivante à notre programme:

```
10,20,30 et 70 comme précédemment
40 WHILE NOT(EOF)
50 LINE INPUT 9,a$(i): i=i+1
60 WEND
```

La boucle WHILE-WEND sera parcourue aussi longtemps qu'il reste des données à lire.

Il peut néanmoins encore se produire un SUBSCRIPT OUT OF RANGE si l'on

tente de lire plus de lignes que n'ont été dimensionnés de string.

Lorsque vous créez des fichiers, je vous conseille de mettre en tête du fichier une variable contenant le nombre de données de l'enregistrement. Quand ultérieurement vous voudrez lire le fichier, il suffira de commencer par aller chercher cette variable, de l'utiliser pour dimensionner le(s) tableau(x), et de poursuivre ensuite la lecture.

Aux adresses &B807 à &B85B se trouvent 32 cases mémoire contenant, sous forme de chaînes ASCII, le nom des fichiers INPUT et OUTPUT. On peut lire le nom du dernier fichier
sauvegardé avec la ligne suivante:

```
FOR i=&B84C TO &B85B: PRINT CHR$(PEEK(i));: NEXT
```

Que pensez vous d'équiper vos programme d'une petite routine qui interdit que l'on modifie le nom ? Il faudrait pour cela qu'en début de programme on lise le nom comme nous venons de le voir; en cas de non conformité le programme serait effacé par un NEW, tout cela agrémenté d'un petit commentaire sarcastique.

Si vous désirez connaître l'actuelle vitesse de chargement, vous pouvez taper PRINT PEEK(&B8D1). Si vous obtenez 6 alors vous êtes en vitesse lente ("Super Prudent"), un 12 signifie charge rapide.

*****

EN BREF: Fichier sur cassette

L'instruction LINE INPUT permet de charger des fichiers contenant des virgules. La fonction EOF indique la fin du fichier. Ces deux instructions permettent de travailler confortablement en BASIC avec des fichiers ASCII en les chargeant dans des tableaux. Le nom du dernier fichier se trouve aux adresses &B807 à &B816 (lecture) et &B84C à &B85B (écriture).
L'octet &B8D1 renseigne sur la vitesse de chargement.

*****

12.2. INKEY VU SOUS UN AUTRE ANGLE

Au plus tard quand vous programmerez votre premier jeu BASIC, vous sentirez la nécessité de pouvoir lire simultanément plusieurs touches. Par chance il se trouve dans le BASIC du CPC une instruction réalisant cette fonction. INKEY(X) (attention: sans le $) indique si la touche X est actuellement enfoncée. Cette fonction ne lit pas le tampon clavier mais teste directement le contact électrique de la touche. Voici un exemple d'application:

```
10 CLS
20 IF INKEY(69)=0 THEN LOCATE 1,5: PRINT "touche A"
30 IF INKEY(36)=0 THEN LOCATE 1,10: PRINT "touche L"
40 LOCATE 1,5: PRINT SPACE$(20): REM effacer ligne 5
50 LOCATE 1,10: PRINT SPACE$(20): REM effacer ligne 10
60 GOTO 20
```

Démarrez le programme et pressez simultanément les touches A et L. Les deux messages s'affichent à l'écran. Avec l'instruction INKEY$ cela

n'aurait pas été possible, celle-ci ne pouvant lire qu'un seul caractère à la fois.

Voyons encore quelques astuces de programmation. Dans de nombreux programmes on trouve une boucle spéciale qui suspend le déroulement du programme jusqu'à ce que n'importe quelle touche soit pressée. Cette boucle a souvent l'allure suivante:

```
WHILE INKEY$="": WEND
```

Il existe en ROM une routine machine assurant la même fonction; on l'appelle par CALL &BB18.

*****

EN BREF: Lecture du clavier

La commande INKEY peut également servir à la lecture simultanée de plusieurs touches.
CALL &BB18 attend qu'une quelconque touche soit pressée.

*****

13. INTRODUCTION AU LANGAGE DU Z-80

Dans de nombreuses publications on trouve des listings de programmes à taper. Bien souvent ceux-ci sont écrits en langage machine, un langage qui plonge les débutants dans la perplexité la plus profonde. Avouons le, l'apprentissage du langage machine n'est pas aussi aisé que le BASIC, en contrepartie il est beaucoup plus rapide et offre au programmeur averti de toutes nouvelles possibilités. C'est pourquoi je voudrais vous présenter ici les bases nécessaires à une programmation dans le langage du Z-80. A la fin de ce chapitre vous serez en mesure de comprendre le principe de fonctionnement de programmes machine, et pourrez alors décider si vous désirez approfondir le sujet. Et si le langage machine ne vous enchante pas, ce chapitre n'aura quand même pas fait de mal, le savoir que vous aurez acquis pourra vous servir pour d'autres langages de programmation. Après tout PASCAL ou LOGO sont aussi d'excellents moyens d'apprendre quelque chose à votre ordinateur.

13.1. MAIS QU'EST-CE DONC QUE LE LANGAGE MACHINE ?

Le langage machine constitue le seul moyen de programmer directement un microprocesseur, sans passer par un interpréteur ou un compilateur. C'est la raison pour laquelle ce langage est aussi prodigieusement rapide.

Le langage machine englobe diverses instructions qui, par des combinaisons adéquates, permettent de réaliser des fonctions

complexes. Grossièrement, on peut diviser les instructions en trois familles. Les plus simples à comprendre pour le programmeur BASIC sont les instructions de branchements, avec lesquelles ont peut sauter allégrement d'une partie du programme à l'autre, tels les GOTO et autres GOSUB. D'autres instructions effectuent des opérations sur les données (par exemple addition, combinaison logique etc.). La dernière famille est constituée par les instructions qui transportent les données d'un endroit de la mémoire à un autre.
Pour le microprocesseur, il n'existe pas de variables. Il ne connaît que les cases mémoire et ses registres internes. Par conséquent il revient au programmeur de distinguer les données du programme lui-même. En général, la manipulation des données ne s'effectue qu'à l'intérieur des registres du Z-80.
Une instruction machine est toujours constituée de ce que l'on appelle un code opératoire (en anglais OPCODE) qui définit la spécificité de l'instruction. La longueur de ce code opératoire peut atteindre jusqu'à 3 octets. De plus, l'opcode peut être suivi d'un ou deux octets de données. De façon purement théorique, les instructions du Z-80 pourraient s'étendre sur jusqu'à 5 octets. En pratique les plus longues comportent 4 octets, les codes opératoires de 3 octets ne pouvant être suivis de plus d'un octet de donnée.

13.2. L'HORLOGE

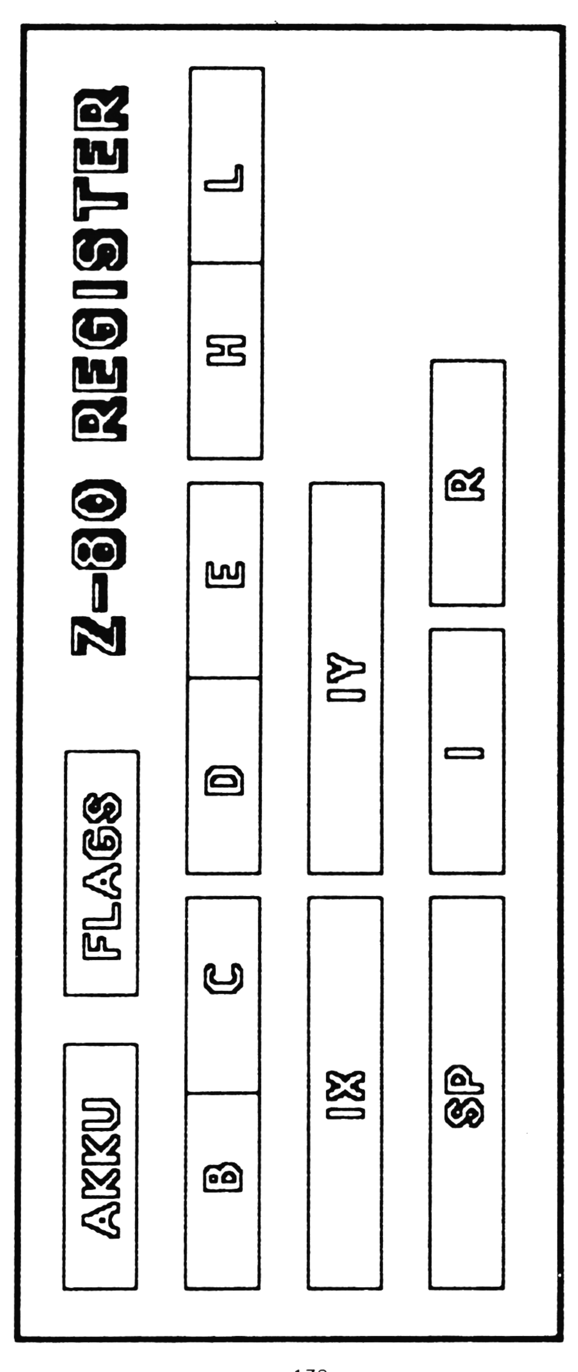
Tous les composants du CPC fonctionnent à la cadence d'un petit quartz insignifiant, qui fournit le signal d'horloge (4 Mégahertz = 4000000 pulsations ou cycles par secondes). Ce signal est nécessaire pour synchroniser les différents circuits. Sans synchronisation il pourrait

par exemple se produire qu'un composant envoie au microprocesseur une donnée alors que celui-ci n'est pas encore prêt à la recevoir. Même le plus rapide des microprocesseurs à besoin d'un petit laps de temps pour exécuter ses instructions.

13.3. CONSTRUCTION DU Z-80

Chaque microprocesseur possède des registres internes dans lesquels sont effectuées les opérations. Le plus important de ces registres est l'accumulateur. C'est là que se déroule la majorité des opérations arithmétiques et logiques. L'accumulateur (abréviation "accu" ou même "a") est un registre à 8 bits et peut donc traiter un octet (en fait, l'accumulateur lui-même ne traite rien, il se contente de recevoir le résultat des opérations). La plupart des instructions arithmétiques nécessitent deux opérandes (par exemple l'addition de deux nombres). Le premier se trouve déjà dans l'accu, le deuxième provient d'un autre registre ou d'une case mémoire. Le résultat de l'opération est remis dans l'accu, il prend ainsi la place de l'opérande s'y trouvant avant.
Un autre registre portant le nom "F" contient les divers flags ("drapeau" en français) qui renseignent sur certains états du microprocesseur. Grâce à ces flags il est par exemple possible de déterminer si le contenu de l'accumulateur est 0.
Mais ce n'est pas tout. Il existe 6 autres registres à 8 bits avec la particularité que, pris deux par deux, ceux-ci constituent des registres à 16 bits. La paire HL peut presque être considérée comme un accumulateur 16 bits car il permet les mêmes opérations que l'accu, mais cette fois sur 16 bits. La manipulation de nombres plus grands

est ainsi grandement simplifiée.
BC et DE constituent deux autres paires de registres. IX et IY sont deux registres d'index (chacun sur 16 bits) qui pointent sur des adresses mémoire. Au moyen de ces pointeurs, il est très facile de manipuler des paquets de données. Je vous expliquerai plus tard comment cela fonctionne.
Le registre 16 bits SP a une fonction toute spéciale. Il pointe toujours sur l'élément se trouvant au sommet de la pile (SP signifie Stack Pointer = pointeur de pile). A chaque fois que l'on ajoute ou retire quelque chose de la pile, le Z-80 actualise ce pointeur pour qu'il indique le nouveau sommet.
Pour en terminer, il existe les registres I et R dont l'usage est réservé à la commande de circuits externes.
Comme si tout cela ne suffisait pas, chacun des registres A et L possèdent un double qui peut prendre la place de l'original. Mais il n'est toujours possible que de travailler dans une seule des paires de registres, c'est pourquoi ces registres servent souvent de mémoire auxiliaire. Les instructions les concernant se reconnaissent à l'apostrophe qui termine le code opératoire.
Voilà, nous venons de faire connaissance avec les registres du Z-80 qui se tiennent à votre disposition pour la programmation en langage machine (voir figure). Dans le chapitre suivant, nous allons suivre l'exécution d'une instruction au sein du microprocesseur.


Z-80 REGISTER
AKKU
FLAGS
B
C
D
E
H
L
IX
IY
SP
I
R

13.4. LE FONCTIONNEMENT DU Z-80

Supposons que dans la mémoire de votre CPC se trouve un programme qui n'attend plus que vous pour commencer l'exécution. Il est évident que le microprocesseur doit connaître l'emplacement du programme. A cet effet il existe un registre 16 bits spécial (encore un !) dans le Z-80, appelé compteur ordinal (en anglais Program Counter = PC). C'est lui qui contient l'adresse de la prochaine instruction à exécuter.
Le microprocesseur va chercher l'octet se trouvant à l'adresse contenue dans le PC, puis l'incrémente de 1 pour qu'il pointe sur l'adresse de l'octet suivant. Pendant ce temps il a décodé le code opératoire (l'octet s'était lui), c.à.d. que le Z-80 reconnaît laquelle des nombreuses instructions dont il s'agit. Certaines instructions sont longues de plusieurs octets (sinon il ne pourrait en exister plus de 256). Dans ce cas les différents octets sont simplement lus à la queue-leu-leu, ceci ne posant pas de problème puisque après chaque lecture le PC avance de 1 octet. Le Z-80 peut également tomber sur des données qu'il ne décodera pas mais qu'il rangera dans certains de ces registres internes; ou bien il les traitera directement.
Si les données doivent subir une quelconque opération (par exemple une addition) c'est maintenant qu'elle a lieu, et le résultat est à nouveau stocké quelque part, par exemple dans l'accu. L'instruction achevée, il s'attaque à la suivante.

Aussi rapide que cela puisse paraître, le déroulement de ces opérations n'est pas instantané; même le courant électrique emploie un certain temps pour parcourir les circuits. En général, chacune des phases "aller chercher code opératoire", "décoder code opératoire",

"exécuter instruction" et "stocker résultat" s'effectue en une période de l'horloge. C'est pourquoi on définit souvent un "cycle machine" comme une suite de 4 pulsation de l'horloge. Des instructions plus complexes peuvent durer plusieurs cycles machine.

*****

EN BREF

Le PC pointe toujours sur l'adresse du prochain octet à traiter. Les opcodes et les données sont lus les uns après les autres. Les codes opératoires sont d'abord décodés puis l'instruction exécutée.

*****

13.5. LE SYSTEME HEXADECIMAL

Toutes les fois que vous aurez à faire avec le langage machine, vous tomberez sur la représentation hexadécimale des nombres. Contrairement à notre système décimal bien connu qui en possède 10, le système hexadécimal est constitué de 16 symboles différents (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). La conversion entre nombres binaires et nombres hexadécimaux est vraiment très simple, c'est pourquoi ce système est tant utilisé. Cela tient entre autre au fait qu'un chiffre hexa correspond exactement à un demi octet (le plus grand nombre hexa à deux chiffre, FF, est égal à la valeur maximale que peut prendre un octet: 1111 1111). La conversion se fait toujours en divisant les octets en deux et en transformant chaque moitié en un chiffre hexadécimal. Voici les correspondances entre nombres décimaux,

hexadécimaux et binaires:

Déc	Hexa	Bin
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Le nombre 1010 1011 (base 2) s'écrit donc AB en hexadécimal, car 1010 (base 2) = A (base 16) et 1011 (base 2) = B (base 16). Cela marche bien sûr aussi dans l'autre sens.

Pour le sens hexa vers décimal, on convertit d'abord chaque chiffre en son équivalent décimal. Le chiffre le plus à droite (les unités) est ensuite multiplié par 16p0 (=1), le deuxième par 16p1 (=16), le troisième par 16p2 (=256) et ainsi de suite. On fait enfin la somme de tous ces produits.

Un exemple:

&ABCD (les chiffres correspondent à 10, 11, 12 et 13)
= 10*16p3 + 11*16p2 + 12*16p1 + 13*16p0
= 10*4096 + 11*256 + 12*16 + 13*1
= 43981

Voici maintenant une méthode pour faire la conversion inverse. On commence par diviser le nombre décimal par 16; on note le reste de la division sous forme hexadécimale. Le résultat de la division est à nouveau divisé par 16, et ainsi de suite jusqu'à obtenir 0. Un exemple nous aidera encore une fois à y voir plus clair:

```
 53000 / 16 = 3312 reste  8 -  8
  3312 / 16 =  207 reste  0 -  0
   207 / 16 =   12 reste 15 -  F
    12 / 16 =    0 reste 12 -  C
=   53000 = &CF08
```

Il existe maintenant des calculatrices équipées de fonctions permettant la conversion des nombres en bases différentes. De bons programmes assembleurs ou moniteurs possèdent également cette possibilité.

Votre CPC aussi sait le faire, il a en effet la faculté de travailler avec les nombres hexadécimaux. Il suffit de faire précéder le nombre d'un & (plus exactement &H), et déjà celui-ci est considéré comme étant de base 16.

13.6. LE CALCUL BINAIRE

13.6.1. ADDITION

Pour tout de suite vous rassurer: l'addition binaire fonctionne exactement comme l'addition décimale.

La somme de deux zéros, ou d'un zéro et d'un "un", ne devrait pas vous poser de problème, on additionne bêtement. Par contre les ennuis commencent lorsque nous voulons calculer 1 + 1. En effet le 2 n'existe pas dans le système binaire. Comme lorsque l'on fait 9 + 1 dans notre système décimal, le résultat est 0 et on a une retenue qui vient s'ajouter au rang suivant:

```
  0     0     1     1
+ 0   + 1   + 0   + 1
---   ---   ---   ---
  0     1     1    10
```

Des octets entiers peuvent ainsi être additionnés bit à bit, en tenant compte à chaque fois d'une éventuelle retenue:

```
  01101101 = 109
+ 00001001 = + 9
+    1 1       (les retenues)
---------- ---
  01110110 = 118
```

Pour que vous voyiez bien le fonctionnement, j'ai représenté les

retenues.

Il peut arriver que l'on ait à additionner deux "1" auxquels vient s'ajouter une retenue. Le résultat est dans ce cas 11 (1+1+1=3 - 11 en binaire), c'est simple, non ?

Essayez un peu de faire le calcul suivant:

```
   10010011
 + 11011111
 +1  111111  (retenues)
 ----------
  101110010
```

Et nous nous retrouvons avec 9 bits ! Ce neuvième bit est pris en compte par le microprocesseur et s'appelle la CARRY. Elle indique qu'il y a eu un dépassement (résultat supérieur à 255 = &FF) et nous amène tout naturellement à l'addition sur 16 bits.

Aucun ordinateur ne peut se contenter de faire ses calculs sur 8 bits, les nombres se trouvent en général dans un domaine bien plus grand. Pourtant les microprocesseurs 8 bits (Z-80, 6502, 6510, 6809...) ne peuvent digérer que 8 bits à la fois. C'est ici que la carry va nous être d'un grand secours. Supposons que nous voulions additionner deux nombres s'étendant chacun sur 2 octets. Il suffit au microprocesseur d'additionner séparément les deux groupes de deux octets et de rajouter au deuxième la retenue (carry) éventuellement survenue à la première opération.

Un exemple:

```
   00110101 10010011
 + 10011011 11011111
 +  1111111   11111  (retenues)
 -------------------
```

11010001 01110010

Vous reconnaissez à droite l'addition de tout à l'heure.

13.6.2. SOUSTRACTION

Pour soustraire un nombre d'un autre, l'ordinateur commence par prendre l'opposé du premier (c.à.d. qu'il le multiplie par -1) puis l'additionne à l'autre. Il procède ainsi par ce qu'il est possible de réaliser une addition avec des portes logiques (AND, OR, XOR, NOT), mais pas une soustraction.
Pour pouvoir représenter les nombres négatifs, le domaine d'un octet à été décalé de 0..255 à -127..+127. Le bit le plus à gauche (MSB=Most Significant Bit=bit de poids fort) sert de signe. Un 1 signifie que le nombre est négatif, un octet commençant par 0 est positif. Mais attention, rendre négatif un nombre ne consiste pas simplement à mettre à 1 le bit 7. Un exemple va nous montrer la difficulté:

```
  00000001
+ 10000001
----------
  10000010
```

Ce qui voudrait dire que 1 + -1 = -2. Ceci est évidemment faux et on a choisi une autre solution. On rend négatif un nombre en formant ce que l'on appelle le complément à deux. Voici comment on
procède: On inverse tous les bits et on ajoute 1 au nombre obtenu.

Exemple:

```
Soit le nombre 91:      01011011
Calculons -91:
 On complémente à 1:    10100100
                      +        1
                      ----------
                        10100101
```

En suivant ce schéma, 1 - 1 se calcule de la manière suivante, et on obtient le bon résultat:

```
     00000001
   + 11111111
    11111111  (retenues)
   ----------
    100000000
```

Apparemment nous obtenons une retenue. Mais ici encore la soustraction se comporte différemment. Dans le cas présent nous pouvons l'ignorer. Si voulions soustraire 16 bits, la carry veillerait à ce que les bits du deuxième octet soient également mis à 0. Le nombre -1 donne 11111111 11111111 sur 16 bits. Sans la carry on aurait obtenu 11111111 00000000, ce qui est faux.
Heureusement le programmeur n'a pas à se soucier de tout cela. Le complément à deux est automatiquement réalisé par les instructions de soustraction.

13.6.3. MULTIPLICATION

Aussi étonnant que cela puisse paraître, le Z-80 ne connaît que deux opérations: l'addition et la soustraction ! Toutes les autres opérations sont réalisées à l'aide de ces deux-là et aussi à l'aide d'autres instructions que nous verrons plus tard.
Etant donné que nous ne voulons pas ici entrer trop dans le détail (il existe une abondante littérature sur ce sujet), je ne vous présenterai que la manière la plus simple d'effectuer une multiplication. Les "professionnels" n'aiment pas l'employer car elle n'est pas très rationnelle.
Pour faire le produit X * n, il suffit d'additionner n fois X. Cela ne fonctionne évidemment que si n est un entier. Dans le cas contraire, il existe des procédés plus complexes, mais qui fonctionnent sur un principe analogue.
Un exemple pour mieux comprendre:

4 * 3 = 4 + 4 + 4 = 12

13.6.4. DIVISION

Une méthode aussi simple que la précédante peut être employée pour faire une division (mais ce n'est pas la seule). Pour diviser X par n, il suffit de soustraire successivement n de X. Le nombre de soustractions nécessaires pour obtenir un nombre inférieur à n donne le résultat de la division.
Voici un exemple:

10 / 3 = ?

10 - 3 = 7   compteur=1
 7 - 3 = 4   compteur=2
 4 - 3 = 1   compteur=3

=> 10 / 3 = 3 reste 1

On peut se permettre de telles longueurs parce que le langage machine est aussi extraordinairement rapide. Signalons que les calculatrices fonctionnent sur le même principe.
A partir des 4 opérations de base, on peut réaliser des opérations plus complexes, comme par exemple les calculs de puissances, les sinus, etc. On voit par conséquent que n'importe quelle opération mathématique peut être réalisée à l'aide des simples opérateurs logiques AND, OR, XOR et NOT (puisque addition et soustraction en sont elles-mêmes une combinaison).

13.7. LES COMPARAISONS, COMMENT CA MARCHE ?

Les comparaisons sont monnaie courante en BASIC. Mais comment procéder en langage machine ? Voyons pour cela l'équivalence suivante:

A = B   =   A - B = 0

Vous voyez qu'il est très facile de modifier une comparaison. Pour l'ordinateur la nouvelle forme a l'avantage de comporter un 0 à droite de l'égalité. Et justement le 0 est le seule nombre dont un

microprocesseur puisse détecter la présence dans son registre de calcul (en général l'accu). Il lui suffit de combiner tous les bits avec OU, à la manière suivante:

b7 OR b6 OR b5 OR b4 OR b3 OR b2 OR b1 OR b0

Le résultat ne vaut 0 que si tous les bits de l'accu étaient à 0. Voilà, nous venons de voir nos deux premières comparaisons: Pour savoir si A = B ou si A   B, il suffit de les soustraire l'un de l'autre et de regarder si le résultat vaut 0 ou non; cette dernière chose se fait à l'aide du bit Z (Z comme Zéro) du registre d'état
F. S'il est à 1, le résultat de la dernière opération effectuée était 0, et si Z=0 le résultat était différent de 0. Ce "flag" n'est pas réservé au seul usage de l'accumulateur. D'un autre côté, toutes les instructions ne modifient pas les flags. Pour le savoir vous pouvez vous reporter à la liste d'instructions en annexe. Mais revenons aux comparaisons.
Avec "supérieur" et "inférieur" nous procédons presque de la même manière que pour "égal". Après la soustraction, on regarde si le nombre qui se trouve dans l'accu est positif ou négatif, indiqué par le bit 7 (voir chapitre 13.6.3.):

A   B   =   A - B   0   =   b7 = 0
A   B   =   A - B   0   =   b7 = 1

Beaucoup d'instructions modifient le bit S de F (S comme Signe).
En testant ce bit on peut déterminer si le résultat de la dernière opération à été négatif ou positif.
Un autre flag s'appelle P/V (étant donné que nous n'aurons pas beaucoup à faire à lui dans ce livre, je le noterai juste P). Il a deux fonctions. Tout d'abord il peut indiquer la parité d'un nombre,

et ensuite il signal si le bit de signe a été modifié incorrectement. Nous ne y intéresserons pas plus car nous voulions seulement jeter un oeil dans le langage machine.

13.8. LE PREMIER PROGRAMME

Après avoir fait connaissance avec les bases nécessaires à toute programmation en langage machine, nous allons pouvoir nous attaquer au programme le plus simple qui soit, un programme qui effectue une addition sur 8 bits.

Il nous faut d'abord communiquer au programme les deux nombres à additionner. Une instruction du type INPUT n'existe pas en langage machine, nous allons donc stocker nos deux valeurs quelque part dans la mémoire de l'ordinateur. Le programme pourra aller les chercher à cet endroit. Nous voici donc arrivé à la première tâche à accomplir. L'instruction "LD A,(nnnn)" fait presque la même chose qu'un PEEK, il va chercher l'octet se trouvant à l'adresse nnnn et le range dans l'accumulateur.

Nous connaissons également l'emplacement du deuxième octet, mais nous ne pouvons malheureusement pas le charger lui aussi dans le microprocesseur avec "LD B,(nnnn)", cette instruction n'existant pas. Il est par contre possible d'utiliser la paire HL comme pointeur sur l'adresse du deuxième octet. Nous chargeons pour cela, avec "LD HL,nnnn", l'adresse dans les registres souhaités. Remarquez que nnnn ne se trouve plus entre parenthèses, indiquant que l'on charge directement la valeur nnnn et non plus l'octet se trouvant à l'adresse

nnnn.

La prochaine instruction additionne enfin les deux valeurs. Elle s'intitule "ADD A,(HL)" et à pour effet de faire la somme des nombres contenus d'une part dans l'accumulateur et d'autre part à l'adresse pointée par le registre HL. Le résultat est remis dans l'accu. Si le résultat est supérieur à 255, le Z-80 l'indique en mettant à 1 la carry.

Etant donné que cela ne nous sert pas à grand chose que le résultat se trouve dans A, nous allons avec l'instruction suivante le déposer à une adresse mémoire d'où il sera possible de le lire avec un PEEK. Cette fonction est réalisée par "LD (nnnn),A". Elle fonctionne exactement comme "LD A,(nnnn)", mais dans l'autre sens.

Finalement un RET termine le sous-programme, comme le ferait un RETURN en BASIC. Il faut savoir pour cela que l'interpréteur considère les appels de programmes machine (avec
l'instruction CALL) exactement comme des sous-programmes machine, il faut par conséquent que la dernière instruction d'une routine en langage machine soit toujours RET, sinon l'ordinateur se plante.

Voilà, nous venons d'écrire allégrement notre premier programme en langage machine, mais nous ne nous sommes pas encore inquiété de son emplacement en mémoire. Dans notre exemple on peut choisir n'importe quelle adresse, il faut seulement veiller à ce que la place ne soit pas déjà utilisée par l'ordinateur lui-même. Je vous conseille de limiter (avec la commande MEMORY) la mémoire BASIC à &AAFF. Vous avez alors à votre disposition tous les octets entre &AB00 et &AB7F.
Nous pouvons maintenant placer notre programme à l'adresse &AB00, les valeurs seront stockées à la fin du domaine que nous venons de

réserver. Le programme ressemblera alors à ceci:

adresse	instruction	commentaire
AB00	LD A,(AB7F)	charger 1ère valeur (ad. AB7F)
AB03	LD HL,(AB7E)	pointer sur 2ième valeur
AB06	ADD A,(HL)	A=A+2ième valeur
AB07	LD (AB7D),A	stocker résultat
AB0A	RET	fin de sous-programme

On peut voir à la valeur des adresses que les instructions prennent plus ou moins d'octets de place. Les instructions comportant une indication d'adresse occupent au moins 3 octets, d'autres se contentent d'un seul (ADD A,(HL)).

Il ne nous reste plus qu'à connaître les codes opératoires des diverses instructions (voir tableau en annexe), et nous pouvons POKEr les octets en mémoire. "LD A,(nnnn)" par exemple s'écrit 3A plus les deux octets d'adresse. Attention, l'octet de poids faible se trouve toujours avant l'octet de poids fort. Pour cette instruction nous obtenons donc:

```
3A 7F AB
```

Voici les codes du programme au grand complet:

```
3A 7F AB     LD A,(AB7F)
21 7E AB     LD HL,AB7E
86           ADD A,(HL)
32 7D AB     LD (AB7D),A
C9           RET
```

Le petit programme BASIC qui suit réserve la mémoire et POKE les valeurs aux adresses correspondantes:

```
10 MEMORY &AAFF
20 FOR i=&AB00 TO &AB0A: READ a: POKE i,a: NEXT
30 DATA &3A,&7F,&AB,&21,&7E,&AB,&86,&32,&7D,&AB,&C9
```

La routine machine peut être lancée avec un CALL &AB00. Auparavant il faut lui fournir les deux valeurs à additionner: POKE &AB7F,n1 et POKE &AB7E,n2. Après le CALL un PRINT PEEK(&AB7D) vous donne le résultat de l'addition. Pour vous permettre de tester confortablement cette première routine, vous pouvez ajouter les lignes suivantes au programme ci-dessus:

```
40 INPUT "n1, n2";n1,n2
50 POKE &AB7F,n1: POKE &AB7E,n2
60 CALL &AB00: PRINT PEEK(&AB7D)
70 GOTO 40
```

La programmation de la soustraction se fait de manière identique, il suffit de remplacer l'instruction ADD A,(HL) par SUB (HL); modifiez pour cela le &86 de la ligne de DATAs en un &96.

13.9. COMMENT PROGRAMMER UNE BOUCLE ?

Lorsque nous voulons en BASIC que certaines instructions soient exécutées plusieurs fois, deux possibilités se présentent à nous:

FOR-NEXT ou WHILE-WEND. Il existe une troisième solution moins élégante et donc moins utilisée. On peut définir une variable comme compteur en l'incrémentant de 1 à chaque passage, puis on revient éventuellement au début au moyen de IF-THEN si le compteur n'a pas atteint une certaine valeur. Aussi compliqué que puisse paraître cette façon de faire, c'est la seule possibilité que nous offre le langage machine. A la place d'une variable, nous utiliserons un des registres du Z-80 et la boucle sera décomptée, en partant d'une valeur donnée. Sinon tout fonctionne comme nous venons de le voir. Le programme qui suit ne fait que parcourir 255 fois la boucle, sans rien faire d'autre (la boucle est vide). Je ne vous propose pas de chargeur BASIC, car l'action du programme n'est pas visible. De plus le langage machine est tellement rapide qu'il parcourt la totalité des 255 boucles en une fraction de seconde.

```
AB00  LD B,FF     charger nombre de passages de la boucle
AB02  DEC B       décrémenter B de 1
AB03  JP NZ,AB02  si   0, saute en AB02
AB06  RET         fin du sous-programme
```

La première instruction charge B avec le nombre de passages à effectuer (ici FF = 255). Le registre B est particulièrement bien adapté à cette fonction de comptage, tout comme A est bien adapté au calcul.

Avec le DEC on soustrait 1 du contenu de B. Si le résultat était 0 le bit Z est mis à 1, sinon celui-ci reste à 0. L'instruction suivante va justement se baser sur Z pour décider ce qu'il va faire. JP NZ ne sautera à l'adresse indiquée qu'à la condition que le bit Z soit à 0, c.à.d. si B n'est pas encore arrivé à 0. Si Z = 1, JP NZ ne fait rien et on passe à l'instruction suivante. Ici c'est RET qui sort du

programme.

13.10. D'AUTRES ROUTINES DE CALCUL

13.10.1. ADDITION SUR 16 BITS

Le Z-80 est un des rares microprocesseurs 8 bits pouvant traiter des nombres à 16 bits. Il possède pour cela les registres 16 bits nécessaires. Le nombre total de valeurs représentables passe alors de 256 (2p8) à 32767 (2p16).

Pour effectuer une addition sur 16 bits, il faut charger le premier nombre dans la paire de registres DE: "LD DE,(nnnn)" met dans D l'octet se trouvant à l'adresse nnnn, et dans E l'octet de l'adresse nnnn+1. Nous retrouvons ici l'ordre Low-High. La même instruction existe pour la paire HL.
L'addition proprement dite s'obtient avec "ADD HL,DE" qui, après avoir additionné DE et HL, remet le résultat dans HL. De là on peut le ramener en mémoire avec "LD (nnnn),HL".
Voici le programme complet:

```
AB00  LD DE,(AB7E)
AB04  LD HL,(AB7C)
AB07  ADD HL,DE
AB08  LD (AB7A),HL
AB0B  RET
```

Rappelez-vous que chacune des deux instructions LD charge deux octets en même temps. Voici à nouveau un chargeur BASIC:

```
10 DATA &ED,&5B,&7E,&AB
11 DATA &2A,&7C,&AB
12 DATA &19
13 DATA &22,&7A,&AB
14 DATA &C9
20 FOR I=&AB00 TO &AB0B: READ A: POKE I,A: NEXT
30 INPUT "n1, n2";n1,n2
40 POKE &AB7E,n1 AND 255: POKE &AB7F,INT(n1/256)
50 POKE &AB7C,n2 AND 255: POKE &AB7D,INT(n2/256)
60 CALL &AB00: PRINT PEEK(&AB7A)+256*PEEK(&AB7B)
70 GOTO 30
```

13.10.2. MULTIPLICATION

Nous avons déjà fait connaissance au chapitre 13.6.3 avec un algorithme de multiplication. Transcrit en BASIC, cela donnerait:

```
10 INPUT "n1, n2";n1,n2: r=0
20 FOR i=1 TO n1
30 r=r+n2: NEXT
40 PRINT n1;"*";n2;"=";r
```

Comme vous voyez, le programme ne contient qu'une boucle et une addition, deux choses que nous savons programmer en langage machine.

Quand on multiplie deux nombres à 8 bits, le résultat occupe 16 bits. Nous allons donc modifier comme il faut la routine d'addition sur 16 bits que nous venons d'étudier.
Seule l'instruction d'addition doit se trouver à l'intérieur de la boucle, le reste est exécuté une fois pour toutes. La première instruction (voir le listing) charge le nombre n1 dans l'accumulateur. On veut en fait le mettre dans B, mais comme il n'existe pas d'instruction adéquate, on passe par l'intermédiaire de A, puis on transfère le nombre de A dans B (deuxième instruction). Cette valeur détermine le nombre de fois que la boucle sera parcourue. A chaque passage on ajoute n2 à la paire HL. C'est pourquoi on le charge maintenant dans E (via le même détour).
Etant donné que ADD HL,DE utilise également D (qui ne nous sert à rien ici) il faut le mettre à 0. HL doit lui aussi contenir 0 avant le premier appel, c'est le rôle des deux instructions suivantes.

Nous arrivons à la boucle (en AB0D). Avant chaque nouvelle addition, le registre HL contient le résultat précédent.
Je pense que la suite du programme se passe de commentaires.
Voici le listing:

```
AB00 LD A,(AB7F)    n1
AB03 LD B,A         dans B
AB04 LD A,(AB7E)    n2
AB07 LD E,A         dans E
AB08 LD D,00        0 dans D
AB0A LD HL,0000     0 dans HL
AB0D ADD HL,DE      additionner
AB0E DEC B          B=B-1
AB0F JP NZ,AB0D     si 0 alors retour
AB12 LD (AB7C),HL   stocker résultat
```

```
AB00 RET              retour au BASIC
```

Voici à nouveau un programme permettant de charger la routine ci-dessus à partir du BASIC:

```
10 DATA &3A,&7F,&AB
11 DATA &47
12 DATA &3A,&7E,&AB
13 DATA &5F
14 DATA &16,&00
15 DATA &21,&00,&00
16 DATA &19,&05
17 DATA &C2,&0D,&AB
18 DATA &22,&7C,&AB
19 DATA &C9
20 FOR I=&AB00 TO &AB15: READ A: POKE I,A: NEXT
30 INPUT "n1, n2";n1,n2
40 POKE &AB7F,n1: POKE &AB7E,n2
50 CALL &AB00: PRINT PEEK(&AB7C)+256*PEEK(&AB7D)
60 GOTO 30
```

13.11. QUELQUES ROUTINES UTILES

Dans ce chapitre je voudrais joindre l'utile à l'agréable et vous présenter quelques routines en langages machines qui ne sont pas seulement là pour vous apprendre le maniement du langage machine, mais qui présentent également une utilité pratique.

Commençons par quelque chose de destructif, effacer toute la mémoire. On peut bien sûr obtenir cela en BASIC avec des POKEs et une boucle FOR-NEXT, mais c'est relativement long. Un programme machine est beaucoup plus rapide. Néanmoins il faut signaler une petite restriction. Avec nos connaissances actuelles nous ne pourrons effacer que par paquets de 256 octets.

Un peu comme le SAVE binaire, il faut indiquer à notre programme l'adresse à partir de laquelle on veut effacer, ainsi que le nombre d'octets à annuler.
HL sert de pointeur et est incrémenté à chaque passage de la boucle. Le bon vieux registre B contient à nouveau la longueur de la boucle (et par conséquent le nombre d'octets à effacer). Ensuite on écrit 0 dans l'octet pointé par HL et au besoin, JP NZ,(nnnn) se rebranche en début de boucle.

Le programme pourrait en rester là, mais nous allons l'étoffer encore un peu. A la fin de l'exécution de la boucle, HL contient l'adresse du premier octet qui n'a pas été effacé. Le programme va stocker cette adresse à l'endroit où il va chercher son adresse de départ. Si vous désirez poursuivre l'effacement, il suffit de rappeler la routine avec un CALL.

La case mémoire contenant la longueur de la boucle n'est pas modifiée par le programme, on n'a donc également pas besoin d'y toucher lors d'appels ultérieurs. Si vous voulez par exemple effacer 100 octets par pas de 10, utilisez la suite d'instructions suivante:

```
POKE &AB7F,10: POKE &AB7D,lowbyte: POKE &AB7E,highbyte
FOR i=1 TO 10: CALL &AB00: NEXT
```

Tel quel, cela ne présente pas grand intérêt, on aurait mieux fait de tout effacer d'un coup. Par contre si on inclut d'autres instructions dans la boucle, qui attendent par exemple qu'une touche soit pressée, on peut imaginer un effacement progressif de l'écran.

Encore une remarque sur la longueur de la boucle. Si l'on donne 0 comme valeur, il y aura 256 passages. En effet avant le premier JP (donc avant la première fin possible) B aura été décrémenté. Or pour le Z-80, et pour tout autre microprocesseur 8 bits, 0 moins 1 donnera toujours 255 (regardez pour cela la représentation binaire de 255, et ajoutez 1).

Voici maintenant notre programme:

```
AB00  LD HL,(AB7D)
AB03  LD A,(AB7F)
AB06  LD B,A
AB07  LD (HL),0
AB09  INC HL
AB0A  DEC B
AB0B  JP NZ,AB07
AB0E  LD (AB7D),HL
AB11  RET
```

Et le chargeur BASIC:

```
10 DATA &2A,&7D,&AB
11 DATA &3A,&7F,&AB
12 DATA &47
13 DATA &36,&00
14 DATA &23
```

```
15 DATA &05
16 DATA &C2,&07,&AB
17 DATA &22,&7D,&AB
18 DATA &C9
20 FOR I=&AB00 TO &AB11: READ A: POKE I,A: NEXT
30 INPUT "adresse de départ";d
40 POKE &AB7D,A-INT(A/256): POKE &AB7E,INT(A/256)
50 INPUT "longueur";l
60 POKE &AB7F,l
70 CALL &AB00
80 GOTO 20
```

La deuxième routine que je vous présente permet de recopier des espaces mémoire. Nous allons utiliser une nouvelle instruction qui raccourcira le programme et balaiera la restriction de tout à l'heure: il sera possible de copier plus de 256 octets.

Cette nouvelle instruction s'intitule LDIR. Elle copie, sans intervention de notre part, des blocs mémoire de longueur quelconque. Il faut auparavant charger cette longueur dans la paire de registres BC. De plus il faut fournir les adresses de départ du bloc d'origine (dans HL) et du bloc d'arrivée (dans DE).
Maintenant LDIR peut entrer en action. Il commence par copier l'octet adressé par HL à l'adresse pointée par DE. Ensuite HL et DE sont incrémentés, BC est décrémenté. Tant que BC est différent de 0, le processus recommence. Une seule instruction remplace toute une boucle !

Tout le reste vous est déjà familier. Remarquez que ce programme est

stocké à un emplacement un peu différent du premier, vous pourrez ainsi les faire tourner en même temps.

```
AB20  LD HL,(AB6E)
AB23  LD DE,(AB6C)
AB27  LD BC,(AB6A)
AB2B  LDIR
AB2D  RET
```

```
10 DATA &2A,&6E,&AB
11 DATA &ED,&5B,&6C,&AB
12 DATA &ED,&4B,&6A,&AB
13 DATA &ED,&B0
14 DATA &C9
20 FOR I=&AB20 TO &AB2D: READ A:POKE I,A: NEXT
30 INPUT "bloc d'origine";a
40 POKE &AB6E,a-256*INT(a/256): POKE &AB6F,INT(a/256)
50 INPUT "bloc d'arrivée";b
60 POKE &AB6C,b-256*INT(b/256): POKE &AB6D,INT(b/256)
70 INPUT "longueur du bloc":c
80 POKE &AB6A,c-256*INT(c/256): POKE &AB6B,INT(c/256)
90 CALL &AB20
100 GOTO 30
```

13.12. LES MODES D'ADRESSAGE

Si vous avez déjà jeté un oeil sur la liste des instructions, vous avez certainement remarqué que les opérandes ne sont pas seulement constitués des divers registres A, B, C, etc. On y trouve aussi des parenthèses et autres fioritures de la sorte. La signification d'une instruction comme ADD A,B paraît évidente; on additionne simplement le contenu des registres A et B. Mais que fait donc ADD A,(HL) ?
D'une manière générale vous devez retenir qu'une expression entre parenthèses représente une adresse et que l'on parle de l'octet se trouvant à cette adresse. "(nn)" veut dire "contenu de l'adresse nnnn" (dans l'écriture abrégée, une lettre représente un octet. C'est pourquoi nn = 2 octets = 16 bits = 4 chiffres hexadécimaux). Cette méthode s'appelle adressage absolu.

En l'absence de parenthèses les valeurs sont directement interprétées comme opérandes. ADD A,n additionne l'octet nn (qui se trouve juste après le code opératoire dans le programme, c'est quasiment une constante) avec l'accumulateur. C'est l'adressage direct.

Il existe également un adressage indirect. Indirect parce que l'opérande ne représente pas l'adresse pointant sur l'octet à traiter, mais l'endroit ou se trouve cette adresse. Si le Z-80 tombe sur l'instruction ADD A,(HL), il ira d'abord voir ce qui se trouve à l'adresse pointée par HL. Cette valeur constituera alors elle même l'adresse à laquelle le microprocesseur ira chercher l'octet, qui sera finalement additionné à l'accu. "Drôlement compliqué" me direz vous. "drôlement rusé" se sont dit les concepteurs du Z-80, cette technique permettant de manipuler des ensembles d'octets comme des tableaux.

L'adressage indirect indicé constitue une variante de l'adressage indirect. L'opérande "(IX+d)" signifie qu'il faut d'abord ajouter la constante dd à la valeur du registre IX pour obtenir l'adresse définitive.

Il nous reste à voir l'adressage relatif que l'on ne rencontre qu'en association avec certaines instructions de branchement. Comme pour l'adressage indicé le code opératoire est suivi d'une constante, qui est additionnée au compteur ordinal (PC). Si le bit 7 est à 1, cette valeur est considérée comme négative (ce qui était d'ailleurs également le cas pour l'adressage indirect-indicé), provoquant en saut en arrière. Etant donné qu'après la lecture de la constante le PC pointe déjà sur l'instruction suivante, le domaine de branchement s'étend de 127 pas en arrière à 129 pas en avant.

13.13. LES INSTRUCTIONS DU Z-80

L'étude de la liste d'instructions en annexe montre que celles-ci peuvent être regroupées par familles dont les membres ne se distinguent que par leurs modes d'adressages. Ce sont ces familles que nous allons maintenant passer en revue. Il n'est pas nécessaire d'apprendre par coeur la fonction de chaque instruction, ce chapitre a plutôt une vocation de manuel de référence que vous pourrez consulter à tout moment. Il vous donnera également un aperçu des possibilités du langage du Z-80.

Etant donné que le fonctionnement des instructions est le même quels que soient les opérandes, ceux-ci ont été remplacés par des lettres (x, n, etc.).

ADC A,x

L'opérande x est additionné au contenu de l'accumulateur. De plus une carry éventuelle est également additionnée. Le résultat retourne dans l'accu, une nouvelle retenue étant stockée dans la carry. Après l'exécution, les flags sont positionnés d'après le contenu de l'accu.

ADC HL,x

Cette instruction fonctionne comme la précédente avec la différence que cette fois l'addition se fait sur 16 bits. L'opérande x (un registre 16 bit) est additionné avec la carry à la paire de registres HL. Le résultat retourne dans HL, les flags sont positionnés en fonction de HL.

ADC A,x

L'opérande x est additionné au contenu de l'accumulateur, sans tenir compte de la carry. Par contre celle-ci peut être modifiée, tout comme les autres flags, en fonction du résultat de l'opération.

ADD HL,x

Comme ADC HL,x, mais sans la carry.

ADD IX,x

X est additionné au registre d'index IX, le résultat retourne dans IX et la carry est actualisée. Par contre les autres flags ne sont pas modifiés.

ADD IY,x

Comme précédemment, mais pour le registre d'index IY.

AND x

Une combinaison ET est réalisée entre x et l'accu, le résultat retourne dans l'accu, les flags sont positionnés en fonction du résultat. Le bit de carry passe à 0 (AND ne produit pas de retenue).

BIT n,x

Le n-ième bit de l'opérande x est testé, c.à.d. que son complément est stocké dans la carry. Si le bit valait 1, on a z=0, et inversement. Les flags S et P prennent des valeurs aléatoires !

CALL nn

Appelle le sous-programme situé à l'adresse nnnn (ressemble au GOSUB du BASIC).

CALL condition,nn

Le sous-programme nnnn n'est appelé que si la condition est réalisée. La condition peut être le test d'un des 4 flags S, Z, P et C(arry). Suivant l'état du flag, le sous-programme sera soit appelé, ou alors on passe à la suite.

CCF

Complémente le bit de carry (1 - 0 et 0 - 1).

CP x

Compare l'opérande x avec l'accu. Pour cela x est soustrait à A, mais le résultat n'est pas sauvegardé, laissant intact l'accumulateur. Les flags sont positionnés en fonction du résultat de la comparaison. Si x est plus petit alors S=1, s'il est supérieur ou égal, S=0; Si les deux valeurs sont égales alors Z=1, sinon Z=0.

CPD

HL sert de pointeur sur une adresse dont le contenu est comparé à l'accumulateur. Si les deux valeurs sont égales alors Z est mis à 1, si l'accumulateur est supérieur ou égal, S=0, et si l'accu est inférieur à l'octet contenu dans l'adresse, S est mis à 1. Ensuite HL et BC sont décrémentés (moins 1). Si BC contient alors 0, le flag P est mis à 0, sinon P=1. Avec cette instruction on se sert de BC comme compteur.

CPDR

Comme CPD, mais l'opération est répétée jusqu'à égalité (Z=1) ou jusqu'à ce que BC=0 (P=0).

CPI

Comme CPD, sauf que HL n'est pas décrémenté mais incrémenté (HL=HL+1).

CPIR

Comme CPDR, mais HL est incrémenté.

CPL

Complémente le contenu de l'accu (par exemple 10011011 devient 01100100).

DAA

Corrige d'éventuels chiffres BCD incorrects (ainsi que les flags), survenus lors d'une opération arithmétique (ADC, ADD, DEC, INC, NEG, SBC, SUB).

DEC x

Décrémente le contenu de x (registre ou adresse), le résultat retourne

dans x. Sauf pour les registres BC, DE, HL, IX, IY et SP, les flags sont positionnés en fonction du résultat.

DI

Masque les interruptions. Après cette instruction, le Z-80 ignore toute demande d'interruption (interruptions masquables).

DJNZ e

B est décrémenté. Si il contient alors autre chose que 0, un branchement relatif est effectué. Pour cela la valeur e est ajoutée au PC (qui se trouve déjà sur l'instruction suivante).

EI

Autorise à nouveau les interruptions.

EX x,y

Echange le contenu des opérandes. Si x est (SP) alors y est échangé avec l'octet se trouvant au sommet de la pile.

EXX

Les registres BC, DE et HL sont échangés avec leurs doubles.

HALT

Le Z-80 interrompt l'exécution jusqu'à la prochaine demande d'interruption.

IM n

Sélectionne le mode d'interruptions.

Mode 0: Le circuit qui envoie la demande d'interruption doit présenter une instruction sur le bus de données.

Mode 1: Le Z-80 se branche à l'adresse &0038 où se trouve une routine

d'interruption. C7est dans ce mode que travaille le CPC. Ne changez jamais de mode, votre ordinateur ne fonctionnerait plus correctement.
Mode 2: Le circuit demandeur fournit la moitié inférieure d'une adresse dont l'autre moitié doit se trouver dans le registre I. A cette adresse doit se trouver une routine d'interruption.

IN x,(C)
Lit dans x un octet venant du port indiqué par le registre C.

IN A,(n)
Lit dans A un octet venant du port n.

INC x
Ajoute 1 à l'opérande x, sinon comme DEC x.

IND
Lit un octet sur le port indiqué par le registre C et le stocke à l'adresse pointée par HL. Puis HL et B sont décrémentés. Si maintenant B=0 alors le flag Z est mis à 1. Les autres flags (exceptée la carry) se positionnent aléatoirement.

INDR
Comme IND, mais l'opération est automatiquement répétée jusqu'à ce que B=0.

INI
Comme IND, mais HL est incrémenté.

INIR
Comme INDR, mais HL est incrémenté.

JP nn

Branchement sans condition à l'adresse nnnn (comme GOTO en BASIC).

JP condition,nn

Effectue un branchement si la condition est réalisée. Les conditions possibles sont les mêmes que pour l'instruction CALL.

JP x

Branchement à l'adresse indiquée par l'opérande x. x peut être (HL), (IX) ou (IY).

JR n et JR condition,n

Comme les instructions JP correspondantes, sauf qu'ici c'est un branchement relatif, c.à.d. que l'octet n est ajouté au PC.

LD x,y

Ceci est la plus grande famille d'instructions. Toutes fonctionnent sur le même principe: le contenu du deuxième opérande est chargé dans le premier. Par exemple LD (nn),A met le contenu de l'accumulateur à l'adresse nnnn. Cela marche aussi bien dans l'autre sens. On peut aussi effectuer des chargement vers ou provenant de registres à 16 bits. Attention si l'on stocke en mémoire le contenu d'un registre à 16 bits, l'octet de poids faible sera rangé à l'adresse nnnn et l'octet de poids fort à l'adresse nnnn+1.

LDD

HL et DE servent de pointeurs sur des adresses mémoire. Le contenu de l'adresse pointée par HL est copié à l'adresse pointée par DE. Ensuite HL, DE et BC sont décrémentés. Si BC=0, alors le flag P est mis à 0. On peut ainsi se servir de BC comme compteur.

LDDR

Comme LDD, mais l'instruction est répétée tant que BC est différent de 0.

LDI

Comme LDD, mais les registres HL et DE sont incrémentés.

LDIR

Comme LDDR, mais HL et DE sont incrémentés.

NEG

Multiplie par -1 le contenu de l'accumulateur. Le résultat retourne dans l'accumulateur et les flags sont actualisés. P=1 si l'accu contenait &80, C=1 s'il contenait 0.

NOP

Cette instruction ne fait rien. Elle sert à programmer des petites pauses (car elle nécessite tout de même un temps d'exécution) ou à remplacer des instructions que l'on a supprimées.

OR x

L'accumulateur et l'opérande subissent une combinaison OU. Le résultat retourne dans l'accu et les flags sont actualisés. La carry est nécessairement mise à 0 car aucune retenue ne peut survenir avec OU.

OTDR

Le contenu de l'adresse pointée par HL est envoyé au port indiqué par le registre B. Ensuite HL et B sont décrémentés. Le processus se répète jusqu'à ce que BC=0. Excepté la carry et le bit Z (qui est mis à 1) tous les flags prennent des états aléatoires.

OTIR

Comme OTDR, mais HL est incrémenté.

OUT (C),x

L'octet se trouvant dans x est envoyé au port dont le numéro est indiqué par le registre C.

OUT (n),A

Le contenu de l'accumulateur est envoyé au port n

OUTD

Le contenu de l'adresse pointée par HL est envoyé au port dont le numéro se trouve dans C. Ensuite HL et B sont décrémentés. Si B=0 alors Z=1, les autre flags (exceptée la carry) obtiennent des valeurs aléatoires.

OUTI

Comme OUTD, mais HL est incrémenté.

POP xy

Va chercher deux registres dans la pile. Le pointeur de pile est actualisé. (AF signifie accumulateur & flags).

PUSH xy

La paire de registres xy est placée sur la pile. Le pointeur de pile est actualisé.

RES n,x

Le n-ième bit de l'opérande x est mis à 0.

RET et RET condition

Ces instructions provoquent un retour de sous-programme (comme RETURN en BASIC). On peut y ajouter une condition, dans ce cas le retour n'aura lieu que si un certain flag se trouve dans un état donné.

RETI

Provoque le retour d'une routine d'interruption.

RETN

Provoque le retour d'une routine NMI.

RL x et RLA

Tous les bits de l'opérande subissent une rotation à gauche, bit 7 est poussé dans la carry qui elle-même se retrouve dans bit 0 (voir figure 9).

L'instruction RLA (contrairement à RL A et aux autres qui peuvent modifier tous les flags) ne modifie que la carry.

C←—7←6←5←4←3←2←1←0←

RL Fig. 9

RLC x et RLCA

L'opérande subit une rotation à gauche. De plus la carry contient bit 7 (voir figure 10).

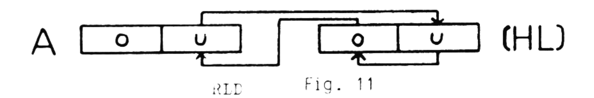
Excepté RLCA les instructions RLC modifient tous les flags.

C←—7←6←5←4←3←2←1←0←

RLC Fig. 10

RLD

Cette instruction effectue une rotation BCD. La moitié gauche (o) de l'octet pointé par HL prend la place de la moitié droite (u) de l'accu, la moitié droite de l'octet va se placer dans la moitié gauche, et l'ancienne moitié droite de l'accu se retrouve dans la partie droite de l'octet (voir figure 11). Les flags S, Z et P sont modifiés.



RLD Fig. 11

RR x et RRA

Comme RL x et RLA, mais cette fois on tourne à droite (figure 12).

↳7→6→5→4→3→2→1→0 ⟶C┘

RR Fig. 12

RRC x et RRCA

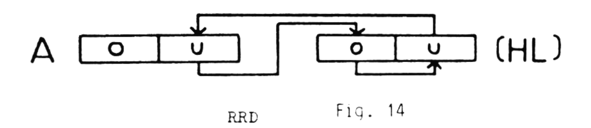
Comme RLC x et RLCA, mais on tourne à droite (figure 13).

↳7→6→5→4→3→2→1→0┴→C

RRC Fig. 13

RRD

Comme RLD avec une rotation à droite. Le nibble (1 nibble = 1/2 octet) inférieur de l'accu est transféré dans le nibble supérieur de l'octet pointé par HL. Le nibble supérieur de celui-ci est décalé dans le nibble inférieur qui lui-même va dans la moitié inférieure de l'accu (voir figure 14).



RRD Fig. 14

RST n

Appelle un sous-programme se trouvant à l'adresse n.

SBC A,x

Le contenu de l'opérande x est soustrait à l'accumulateur en tenant compte d'une éventuelle carry. Le résultat retourne dans l'accu. Les flags sont positionnés en fonction de la nouvelle valeur de l'accumulateur.

SBC HL,x

Comme précédemment, mais sur 16 bits. L'opérande (un registre à 16 bits) ainsi que la carry sont soustraits à la paire HL. Le résultat retourne dans HL et les flags sont actualisés.

SCF

Met la carry à 1.

SET n,x

Le n-ième bit de l'opérande x est mis à 1.

SLA x

Tous les bits de l'opérande sont décalés à gauche. bit 7 se retrouve dans la carry et bit 0 prend la valeur 0 (figure 15). Les flags sont actualisés.

C←—7←6←5←4←3←2←1←0←—(0)

SLA Fig. 15

SRA x

Tous les bits de x sont décalés à droite. Bit 7 (le signe) est conservé et bit 0 se retrouve dans la carry (figure 16). Les flags sont modifiés.

$$\lfloor 7 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 0 \longrightarrow C$$

SRA Fig. 16

SRL x

Tous les bits de x sont décalés à droite. Bit 7 prend la valeur 0 et bit 0 se retrouve dans la carry (figure 17). Les flags sont modifiés.

$$\textcircled{0} \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 0 \longrightarrow C$$

SRL Fig. 17

SUB x

L'opérande x est soustrait à l'accu et le résultat retourne dans l'accu. Les flags sont modifiés.

XOR x

Une combinaison OU-EXCLUSIF est effectuée entre l'opérande et l'accumulateur et le résultat retourne dans l'accumulateur. Les flags sont actualisés (la carry est mise à 0).

13.14. LES CODES OPERATOIRES DU Z-80

Dans le tableau qui suit vous trouverez toutes les instructions du Z-80, avec leur code opératoire correspondant. De plus dans la colonne "flags" sont indiqués les flags modifiés par chaque instruction. Certains flags peuvent prendre des valeurs aléatoires; dans ce cas veuillez vous reporter à la description détaillée de l'instruction.

Mnémonique	Opcode	Flags	Description
ADC A,A	8F	S Z P C	Additionne Car et Accu à A
ADC A,B	88	S Z P C	... à B
ADC A,C	89	S Z P C	... à C
ADC A,D	8A	S Z P C	... à D
ADC A,E	8B	S Z P C	... à E
ADC A,H	8C	S Z P C	... à H
ADC A,L	8D	S Z P C	... à L
ADC A,n	CEnn	S Z P C	... à l'octet n
ADC A,(HL)	8E	S Z P C	... à l'adresse dans HL
ADC A,(IX+d)	DD8Edd	S Z P C	... à l'adresse IX+d
ADC A,(IY+d)	FD8Edd	S Z P C	... à l'adresse IY+d
ADC HL,BC	ED4A	S Z P C	Additionne Car et HL à BC
ADC HL,DE	ED5A	S Z P C	... à DE
ADC HL,HL	ED6A	S Z P C	... à HL
ADC HL,SP	ED7A	S Z P C	... à SP
ADD A,A	87	S Z P C	Additionne Accu à Accu
ADD A,B	80	S Z P C	... à B
ADD A,C	81	S Z P C	... à C
ADD A,D	82	S Z P C	... à D
ADD A,E	83	S Z P C	... à E

ADD A,H	84	S Z P C	... à H
ADD A,L	85	S Z P C	... à L
ADD A,n	C6nn	S Z P C	... à l'octet n
ADD A,(HL)	86	S Z P C	... à l'adresse dans HL
ADD A,(IX+d)	DD86dd	S Z P C	... à l'adresse IX+d
ADD A,(IY+d)	FD86dd	S Z P C	... à l'adresse IY+d
ADD HL,BC	09	C	Additionne HL à BC
ADD HL,DE	19	C	... à DE
ADD HL,HL	29	C	... à HL
ADD HL,SP	39	C	... à SP
ADD IX,BC	DD09	C	Additionne IX à BC
ADD IX,DE	DD19	C	... à DE
ADD IX,HL	DD29	C	... à HL
ADD IX,SP	DD39	C	... à SP
ADD IY,BC	FD09	C	Additionne IY à BC
ADD IY,DE	FD19	C	... à DE
ADD IY,HL	FD29	C	... à HL
ADD IY,SP	FD39	C	... à SP
AND A	A7	S Z P C=0	Combine AND Accu avec Accu
AND B	A0	S Z P C=0	... avec B
AND C	A1	S Z P C=0	... avec C
AND D	A2	S Z P C=0	... avec D
AND E	A3	S Z P C=0	... avec E
AND H	A4	S Z P C=0	... avec H
AND L	A5	S Z P C=0	... avec L
AND n	E6nn	S Z P C=0	... avec l'octet n
AND (HL)	96	S Z P C=0	... avec adresse dans HL
AND (IX+d)	DD96dd	S Z P C=0	... avec l'adresse IX+d
AND (IY+d)	FD96dd	S Z P C=0	... avec l'adresse IY+d
BIT 0,A	CB47	Z	Teste bit 0 de l'accu
BIT 0,B	CB40	Z	Teste bit 0 de B

BIT 0,C	CB41	Z	... de C
BIT 0,D	CB42	Z	... de D
BIT 0,E	CB43	Z	... de E
BIT 0,H	CB44	Z	... de H
BIT 0,L	CB45	Z	... de L
BIT 0,(HL)	CB46	Z	... de l'adresse dans HL
BIT 0,(IX+d)	DDCBdd46	Z	... de l'adresse IX+d
BIT 0,(IY+d)	FDCBdd46	Z	... de l'adresse IY+d
BIT 1,A	CB4F	Z	Teste bit 1 de l'accu
BIT 1,B	CB48	Z	Teste bit 1 de B
BIT 1,C	CB49	Z	... de C
BIT 1,D	CB4A	Z	... de D
BIT 1,E	CB4B	Z	... de E
BIT 1,H	CB4C	Z	... de H
BIT 1,L	CB4D	Z	... de L
BIT 1,(HL)	CB4E	Z	... de l'adresse dans HL
BIT 1,(IX+d)	DDCBdd4E	Z	... de l'adresse IX+d
BIT 1,(IY+d)	FDCBdd4E	Z	... de l'adresse IY+d
BIT 2,A	CB57	Z	Teste bit 2 de l'accu
BIT 2,B	CB50	Z	Teste bit 2 de B
BIT 2,C	CB51	Z	... de C
BIT 2,D	CB52	Z	... de D
BIT 2,E	CB53	Z	... de E
BIT 2,H	CB54	Z	... de H
BIT 2,L	CB55	Z	... de L
BIT 2,(HL)	CB56	Z	... de l'adresse dans HL
BIT 2,(IX+d)	DDCBdd56	Z	... de l'adresse IX+d
BIT 2,(IY+d)	FDCBdd56	Z	... de l'adresse IY+d
BIT 3,A	CB5F	Z	Teste bit 3 de l'accu
BIT 3,B	CB58	Z	Teste bit 3 de B
BIT 3,C	CB59	Z	... de C

BIT 3,D	CB5A	Z	... de D
BIT 3,E	CB5B	Z	... de E
BIT 3,H	CB5C	Z	... de H
BIT 3,L	CB5D	Z	... de L
BIT 3,(HL)	CB5E	Z	... de l'adresse dans HL
BIT 3,(IX+d)	DDCBdd5E	Z	... de l'adresse IX+d
BIT 3,(IY+d)	FDCBdd5E	Z	... de l'adresse IY+d
BIT 4,A	CB67	Z	Teste bit 4 de l'accu
BIT 4,B	CB60	Z	Teste bit 4 de B
BIT 4,C	CB61	Z	... de C
BIT 4,D	CB62	Z	... de D
BIT 4,E	CB63	Z	... de E
BIT 4,H	CB64	Z	... de H
BIT 4,L	CB65	Z	... de L
BIT 4,(HL)	CB66	Z	... de l'adresse dans HL
BIT 4,(IX+d)	DDCBdd66	Z	... de l'adresse IX+d
BIT 4,(IY+d)	FDCBdd66	Z	... de l'adresse IY+d
BIT 5,A	CB6F	Z	Teste bit 5 de l'accu
BIT 5,B	CB68	Z	Teste bit 5 de B
BIT 5,C	CB69	Z	... de C
BIT 5,D	CB6A	Z	... de D
BIT 5,E	CB6B	Z	... de E
BIT 5,H	CB6C	Z	... de H
BIT 5,L	CB6D	Z	... de L
BIT 5,(HL)	CB6E	Z	... de l'adresse dans HL
BIT 5,(IX+d)	DDCBdd6E	Z	... de l'adresse IX+d
BIT 5,(IY+d)	FDCBdd6E	Z	... de l'adresse IY+d
BIT 6,A	CB77	Z	Teste bit 6 de l'accu
BIT 6,B	CB70	Z	Teste bit 6 de B
BIT 6,C	CB71	Z	... de C
BIT 6,D	CB72	Z	... de D

BIT 6,E	CB73	Z	... de E
BIT 6,H	CB74	Z	... de H
BIT 6,L	CB75	Z	... de L
BIT 6,(HL)	CB76	Z	... de l'adresse dans HL
BIT 6,(IX+d)	DDCBdd76	Z	... de l'adresse IX+d
BIT 6,(IY+d)	FDCBdd76	Z	... de l'adresse IY+d
BIT 7,A	CB7F	Z	Teste bit 7 de l'accu
BIT 7,B	CB78	Z	Teste bit 7 de B
BIT 7,C	CB79	Z	... de C
BIT 7,D	CB7A	Z	... de D
BIT 7,E	CB7B	Z	... de E
BIT 7,H	CB7C	Z	... de H
BIT 7,L	CB7D	Z	... de L
BIT 7,(HL)	CB7E	Z	... de l'adresse dans HL
BIT 7,(IX+d)	DDCBdd7E	Z	... de l'adresse IX+d
BIT 7,(IY+d)	FDCBdd7E	Z	... de l'adresse IY+d
CALL nn	CDnnnn		Appelle sous-programme nnnn
CALL C,nn	DCnnnn		... si Carry=1
CALL M,nn	FCnnnn		... si S=1 (négatif)
CALL NC,nn	D4nnnn		... si Carry=0
CALL NZ,nn	C4nnnn		... si Z=0
CALL P,nn	F4nnnn		... si S=0 (positif)
CALL PE,nn	ECnnnn		... si P=1
CALL PO,nn	E4nnnn		... si P=0
CALL Z,nn	CCnnnn		... si Z=1
CCF	3F	C	Complémente Carry-flag
CP A	BF	S Z P C	Compare A avec A
CP B	B8	S Z P C	... avec B
CP C	B9	S Z P C	... avec C
CP D	BA	S Z P C	... avec D
CP E	BB	S Z P C	... avec E

CP H	BC	S Z P C	... avec H
CP L	BD	S Z P C	... avec L
CP n	FEnn	S Z P C	... avec l'octet n
CP (HL)	BE	S Z P C	... avec l'adresse HL
CP (IX+d)	DDBEdd	S Z P C	... avec l'adresse IX+d
CP (IY+d)	FDBEdd	S Z P C	... avec l'adresse IY+d
CPD	EDA9	S Z P	Compare & décrémente
CPDR	EDB9	S Z P	Compare bloc & décrémente
CPI	EDA1	S Z P	Compare & incrémente
CPIR	EDB1	S Z P	Compare bloc & incrémente
CPL	2F		Complémente accu
DAA	27	S Z P C	Adaptation BCD de l'accu
DEC A	3D	S Z P	décrémente accu
DEC B	05	S Z P	... B
DEC BC	0B		... BC
DEC C	0D	S Z P	... C
DEC D	15	S Z P	... D
DEC DE	1B		... DE
DEC DE	1B		... DE
DEC E	1D	S Z P	... E
DEC H	25	S Z P	... H
DEC HL	2B		... HL
DEC IX	D2B		... IX
DEC IY	FD2B		... IY
DEC L	2D	S Z P	... L
DEC SP	3B		... SP
DEC (HL)	35	S Z P	... adresse dans HL
DEC (IX+d)	DD35dd	S Z P	... adresse IX+d
DEC (IY+d)	FD35dd	S Z P	... adresse IY+d
DI	F3		Empêche interruption
DJNZ e	10ee		Décrémente & branche si 0

EI	FB		Autorise interruption
EX AF,AF'	08		Intervertit accu/flags avec leur double
EX DE,HL	EB		Intervertit DE et HL
EX (SP),HL	E3		Intervertit sommet pile/HL
EX (SP),IX	DDE3		... IX
EX (SP),IY	FDE3		... IY
EXX	D9		... BC,DE,HL/doubles
HALT	76		Attend interruption
IM 0	ED46		Mode 0 d'interruptions
IM 1	ED56		Mode 1 d'interruptions
IM 2	ED5E		Mode 2 d'interruptions
IN A,(C)	ED78	S Z P	Lit octet de port C dans A
IN A,(n)	DBnn		... du port n
IN B,(C)	ED40	S Z P	... dans B
IN C,(C)	ED48	S Z P	... dans C
IN D,(C)	ED50	S Z P	... dans D
IN E,(C)	ED58	S Z P	... dans E
IN H,(C)	ED60	S Z P	... dans H
IN L,(C)	ED68	S Z P	... dans L
INC A	3C	S Z P	Incrémente l'accu
INC B	04	S Z P	... B
INC BC	03		... BC
INC C	0C	S Z P	... C
INC D	14	S Z P	... D
INC DE	13		... DE
INC E	1C	S Z P	... E
INC H	24	S Z P	... H
INC HL	23		... HL
INC IX	DD23		... IX
INC IY	FD23		... IY

INC L	2C	S Z P	... L
INC SP	33		... SP
INC (HL)	34	S Z P	... adresse dans HL
INC (IX+d)	DD34dd	S Z P	... adresse IX+d
INC (IY+d)	FD34dd	S Z P	... adresse IY+d
IND	EDAA	Z	Lit sur port et décrémente
INDR	EDBA	Z=1	Lit bloc sur port & décrém.
INI	EDA2	Z	Lit sur port et incrémente
INIR	EDB2	Z=1	Lit bloc sur port & incrém.
JP C,nn	DAnnnn		Saute en nn si C=1
JP M,nn	FAnnnn		... si S=1
JP NC,nn	D2nnnn		... si C=0
JP nn	C3nnnn		... sans condition
JP NZ,nn	C2nnnn		... si Z=0
JP P,nn	F2nnnn		... si S=0
JP PE,nn	EAnnnn		... si P=1
JP PO,nn	E2nnnn		... si P=0
JP Z,nn	CAnnnn		... si Z=1
JP (HL)	E9		... à l'adresse dans HL
JP (IX)	DDE9		... à l'adresse dans IX
JP (IY)	FDE9		... à l'adresse dans IY
JR C,e	38ee		Saut relatif si C=1
JR e	18ee		... en PC+e
JR NC,e	30ee		... si C=0
JR NZ,e	20ee		... si Z=0
JR Z,e	28ee		... si Z=1
LD A,A	7F		Charge accu avec accu
LD A,B	78		... avec B
LD A,C	79		... avec C
LD A,D	7A		... avec D
LD A,E	7B		... avec E

LD A,H	7C		... avec H
LD A,I	ED57	S Z P	... avec I
LD A,L	7D		... avec L
LD A,n	3Enn		... avec octet n
LD A,R	ED5F	S Z P	... avec R
LD A,(BC)	0A		... de l'adresse dans BC
LD A,(DE)	1A		... de l'adresse dans DE
LD A,(HL)	7E		... de l'adresse dans HL
LD A,(IX+d)	DD7Edd		... de l'adresse IX+d
LD A,(IY+d)	FD7Edd		... de l'adresse IY+d
LD A,(nn)	3Annnn		... de l'adresse nn
LD B,A	47		Charge B avec accu
LD B,B	40		... avec B
LD B,C	41		... avec C
LD B,D	42		... avec D
LD B,E	43		... avec E
LD B,H	44		... avec H
LD B,L	45		... avec L
LD B,n	06nn		... avec octet n
LD B,(HL)	46		... de l'adresse dans HL
LD B,(IX+d)	DD46dd		... de l'adresse IX+d
LD B,(IY+d)	FD46dd		... de l'adresse IY+d
LD BC,nn	01nnnn		Charge BC avec octets nn
LD BC,(nn)	ED4Bnnnn		... de nn et nn+1
LD C,A	4F		Charge C avec accu
LD C,B	48		... avec B
LD C,C	49		... avec C
LD C,D	4A		... avec D
LD C,E	4B		... avec E
LD C,H	4C		... avec H
LD C,L	4D		... avec L

LD C,n	0Enn	... avec l'octet n
LD C,(HL)	4E	... de l'adresse dans HL
LD C,(IX+d)	DD4Edd	... de l'adresse IX+d
LD C,(IY+d)	FD4Edd	... de l'adresse IY+d
LD D,A	57	Charge D avec l'accu
LD D,B	50	... avec B
LD D,C	51	... avec C
LD D,D	52	... avec D
LD D,E	53	... avec E
LD D,H	54	... avec H
LD D,L	55	... avec L
LD D,n	16nn	... avec octet n
LD D,(HL)	56	... de l'adresse dans HL
LD D,(IX+d)	DD56dd	... de l'adresse IX+d
LD D,(IY+d)	FD56dd	... de l'adresse IY+d
LD DE,nn	11nnnn	Charge DE avec octets nn
LD DE,(nn)	ED5Bnnnn	... de nn et nn+1
LD E,A	5F	Charge E avec accu
LD E,B	58	... avec B
LD E,C	59	... avec C
LD E,D	5A	... avec D
LD E,E	5B	... avec E
LD E,H	5C	... avec H
LD E,L	5D	... avec L
LD E,n	1Enn	... avec octet n
LD E,(HL)	5E	... de l'adresse dans HL
LD E,(IX+d)	DD5Edd	... de l'adresse IX+d
LD E,(IY+d)	FD5Edd	... de l'adresse IY+d
LD H,A	67	Charge H avec accu
LD H,B	60	... avec B
LD H,C	61	... avec C

LD H,D	62	... avec D
LD H,E	63	... avec E
LD H,H	64	... avec H
LD H,L	65	... avec L
LD H,n	26nn	... avec octet n
LD H,(HL)	66	... de l'adresse dans HL
LD H,(IX+d)	DD66dd	... de l'adresse IX+d
LD H,(IY+d)	FD66dd	... de l'adresse IY+d
LD HL,nn	21nnnn	Charge HL avec octets nn
LD HL,(nn)	2Annnn	... de nn et nn+1
LD I,A	ED47	Charge I avec accu
LD IX,nn	DD21nnnn	Charge IX avec octets nn
LD IX,(nn)	DD2Annnn	... de nn et nn+1
LD IY,nn	FD21nnnn	Charge IY avec octets nn
LD IY,(nn)	FD2Annnn	... de nn et nn+1
LD L,A	6F	Charge L avec accu
LD L,B	68	... avec B
LD L,C	69	... avec C
LD L,D	6A	... avec D
LD L,E	6B	... avec E
LD L,H	6C	... avec H
LD L,L	6D	... avec L
LD L,n	2Enn	... avec octet n
LD L,(HL)	6E	... de l'adresse dans HL
LD L,(IX+d)	DD6Edd	... de l'adresse IX+d
LD L,(IY+d)	FD6Edd	... de l'adresse IY+d
LD R,A	ED4F	Charge R avec accu
LD SP,HL	F9	Charge SP avec HL
LD SP,IX	DDF9	... avec IX
LD SP,IY	FDF9	... avec IY
LD SP,nn	31nnnn	... avec octets nn

LD SP,(nn)	ED7Bnnnn	... de nn et nn+1
LD (BC),A	02	Charge adresse dans BC avec l'accu
LD (DE),A	12	... dans DE avec accu
LD (HL),A	77	... dans HL avec accu
LD (HL),B	70	... avec B
LD (HL),C	71	... avec C
LD (HL),D	72	... avec D
LD (HL),E	73	... avec E
LD (HL),H	74	... avec H
LD (HL),L	75	... avec L
LD (HL),n	36nn	... avec octet n
LD (IX+d),A	DD77dd	Charge adresse IX+d a. accu
LD (IX+d),B	DD70dd	... avec B
LD (IX+d),C	DD71dd	... avec C
LD (IX+d),D	DD72dd	... avec D
LD (IX+d),E	DD73dd	... avec E
LD (IX+d),H	DD74dd	... avec H
LD (IX+d),L	DD75dd	... avec L
LD (IX+d),n	DD36ddnn	... avec octet n
LD (IY+d),A	FD77dd	Charge adresse IY+d a. accu
LD (IY+d),B	FD70dd	... avec B
LD (IY+d),C	FD71dd	... avec C
LD (IY+d),D	FD72dd	... avec D
LD (IY+d),E	FD73dd	... avec E
LD (IY+d),H	FD74dd	... avec H
LD (IY+d),L	FD75dd	... avec L
LD (IY+d),n	FD36ddnn	... avec octet n
LD (nn),A	32nnnn	Charge octet n avec accu
LD (nn),BC	ED43nnnn	... a. C & nn+1 a. B
LD (nn),DE	ED53nnnn	... a. E & nn+1 a. D

LD (nn),HL	22nnnn		... a. H & nn+1 a. L
LD (nn),IX	DD22nnnn		... & nn+1 avec IX
LD (nn),IY	FD22nnnn		... & nn+1 avec IY
LD (nn),SP	ED73nnnn		... & nn+1 avec SP
LDD	EDA8	P	Charge et décrémente
LDDR	EDB8	P=0	Charge bloc & décrémente
LDI	EDA0	P	Charge et incrémente
LDIR	EDB0	P=0	Charge bloc & incrémente
NEG	ED44	S Z P C	Rend accu négatif
NOP	00		Ne fait rien
OR A	B7	S Z P C	Combine OU accu avec accu
OR B	B0	S Z P C	... avec B
OR C	B1	S Z P C	... avec C
OR D	B2	S Z P C	... avec D
OR E	B3	S Z P C	... avec E
OR H	B4	S Z P C	... avec H
OR L	B5	S Z P C	... avec L
OR n	F6nn	S Z P C	... avec octet n
OR (HL)	B6	S Z P C	... avec adresse dans HL
OR (IX+d)	DDB6dd	S Z P C	... avec adresse IX+d
OR (IY+d)	FDB6dd	S Z P C	... avec adresse IY+d
OTDR	EDBB	S Z P	Envoie bloc et décrémente
OTIR	EDB3	S Z P	Envoie bloc et incrémente
OUT (C),A	ED79		Envoie sur port C l'accu
OUT (C),B	ED41		... B
OUT (C),C	ED49		... C
OUT (C),D	ED51		... D
OUT (C),E	ED59		... E
OUT (C),H	ED61		... H
OUT (C),L	ED69		... L
OUT (n),A	D3nn		Envoie accu sur port n

OUTD	EDAB	S Z P	Envoie et décrémente
OUTI	EDA3	S Z P	Envoie et incrémente
POP AF	F1		Charge accu/flags de la pile
POP BC	C1		Va chercher BC sur la pile
POP DE	D1		Va chercher DE sur la pile
POP HL	E1		Va chercher HL sur la pile
POP IX	DDE1		Va chercher IX sur la pile
POP IY	FDE1		Va chercher IY sur la pile
PUSH AF	F5		Dépose accu/flags sur pile
PUSH BC	C5		Dépose BC sur la pile
PUSH DE	D5		Dépose DE sur la pile
PUSH HL	E5		Dépose HL sur la pile
PUSH IX	DDE5		Dépose IX sur la pile
PUSH IY	FDE5		Dépose IY sur la pile
RES 0,A	CB87		Met à 0 bit 1 de l'accu
RES 0,B	CB80		... de B
RES 0,C	CB81		... de C
RES 0,D	CB82		... de D
RES 0,E	CB83		... de E
RES 0,H	CB84		... de H
RES 0,L	CB85		... de L
RES 0,(HL)	CB86		... de l'adresse dans HL
RES 0,(IX+d)	DDCBdd86		... de l'adresse IX+d
RES 0,(IY+d)	FDCBdd86		... de l'adresse IY+d
RES 1,A	CB8F		Met à 0 bit 1 de l'accu
RES 1,B	CB88		... de B
RES 1,C	CB89		... de C
RES 1,D	CB8A		... de D
RES 1,E	CB8B		... de E
RES 1,H	CB8C		... de H
RES 1,L	CB8D		... de L

RES 1,(HL)	CB8E	... de l'adresse dans HL
RES 1,(IX+d)	DDCBdd8E	... de l'adresse IX+d
RES 1,(IY+d)	FDCBdd8E	... de l'adresse IY+d
RES 2,A	CB97	Met à 0 bit 2 de l'accu
RES 2,B	CB90	... de B
RES 2,C	CB91	... de C
RES 2,D	CB92	... de D
RES 2,E	CB93	... de E
RES 2,H	CB94	... de H
RES 2,L	CB95	... de L
RES 2,(HL)	CB96	... de l'adresse dans HL
RES 2,(IX+d)	DDCBdd96	... de l'adresse IX+d
RES 2,(IY+d)	FDCBdd96	... de l'adresse IY+d
RES 3,A	CB9F	Met à 0 bit 3 de l'accu
RES 3,B	CB98	... de B
RES 3,C	CB99	... de C
RES 3,D	CB9A	... de D
RES 3,E	CB9B	... de E
RES 3,H	CB9C	... de H
RES 3,L	CB9D	... de L
RES 3,(HL)	CB9E	... de l'adresse dans HL
RES 3,(IX+d)	DDCBdd9E	... de l'adresse IX+d
RES 3,(IY+d)	FDCBdd9E	... de l'adresse IY+d
RES 4,A	CBA7	Met à 0 bit 4 de l'accu
RES 4,B	CBA0	... de B
RES 4,C	CBA1	... de C
RES 4,D	CBA2	... de D
RES 4,E	CBA3	... de E
RES 4,H	CBA4	... de H
RES 4,L	CBA5	... de L
RES 4,(HL)	CBA6	... de l'adresse dans HL

RES 4,(IX+d)	DDCBddA6	... de l'adresse IX+d
RES 4,(IY+d)	FDCBddA6	... de l'adresse IY+d
RES 5,A	CBAF	Met à 0 bit 5 de l'accu
RES 5,B	CBA8	... de B
RES 5,C	CBA9	... de C
RES 5,D	CBAA	... de D
RES 5,E	CBAB	... de E
RES 5,H	CBAC	... de H
RES 5,L	CBAD	... de L
RES 5,(HL)	CBAE	... de l'adresse dans HL
RES 5,(IX+d)	DDCBddAE	... de l'adresse IX+d
RES 5,(IY+d)	FDCBddAE	... de l'adresse IY+d
RES 6,A	CBB7	Met à 0 bit 6 de l'accu
RES 6,B	CBB0	... de B
RES 6,C	CBB1	... de C
RES 6,D	CBB2	... de D
RES 6,E	CBB3	... de E
RES 6,H	CBB4	... de H
RES 6,L	CBB5	... de L
RES 6,(HL)	CBB6	... de l'adresse dans HL
RES 6,(IX+d)	DDCBddB6	... de l'adresse IX+d
RES 6,(IY+d)	FDCBddB6	... de l'adresse IY+d
RES 7,A	CBBF	Met à 0 bit 7 de l'accu
RES 7,B	CBB8	... de B
RES 7,C	CBB9	... de C
RES 7,D	CBBA	... de D
RES 7,E	CBBB	... de E
RES 7,H	CBBC	... de H
RES 7,L	CBBD	... de L
RES 7,(HL)	CBBE	... de l'adresse dans HL
RES 7,(IX+d)	DDCBddBE	... de l'adresse IX+d

RES 7,(IY+d)	FDCBddBE		... de l'adresse IY+d
RET	C9		Retour de sous-programme
RET C	D8		Retour de ss-pg si C=1
RET M	F8		... si S=1
RET NC	D0		... si C=0
RET NZ	C0		... si Z=0
RET P	F0		... si S=0
RET PE	E8		... si P=1
RET PO	E0		... si P=0
RET Z	C8		... si Z=1
RETI	ED4D		Retour d'interruption
RETN	ED45		... de routine NMI
RL A	CB17	S Z P C	Rotation à gauche carry/accu
RL B	CB10	S Z P C	... carry & B
RL C	CB11	S Z P C	... et C
RL D	CB12	S Z P C	... et D
RL E	CB13	S Z P C	... et E
RL H	CB14	S Z P C	... et H
RL L	CB15	S Z P C	... et L
RL (HL)	CB16	S Z P C	... et l'adresse dans HL
RL (IX+d)	DDCBdd16	S Z P C	... et l'adresse IX+d
RL (IY+d)	FDCBdd16	S Z P C	... et l'adresse IY+d
RLA	17	C	... et l'accu
RLC A	CB07	S Z P C	Rotation à gauche de l'accu
RLC B	CB00	S Z P C	... de B
RLC C	CB01	S Z P C	... de C
RLC D	CB02	S Z P C	... de D
RLC E	CB03	S Z P C	... de E
RLC H	CB04	S Z P C	... de H
RLC L	CB05	S Z P C	... de L
RLC (HL)	CB06	S Z P C	... de l'adresse dans HL

RLC (IX+d)	DDCBdd06	S Z P C	... de l'adresse IX+d
RLC (IYd)	FDCBdd06	S Z P C	... de l'adresse IY+d
RLCA	07	C	... de l'accu
RLD	ED6F	S Z P	Rotation décimale à gauche
RR A	CB1F	S Z P C	Rotation à droite carry/accu
RR B	CB18	S Z P C	... carry et B
RR C	CB19	S Z P C	... et C
RR D	CB1A	S Z P C	... et D
RR E	CB1B	S Z P C	... et E
RR H	CB1C	S Z P C	... et H
RR L	CB1D	S Z P C	... et L
RR (HL)	CB1E	S Z P C	... et l'adresse dans HL
RR (IX+d)	DDCBdd1E	S Z P C	... et l'adresse IX+d
RR (IY+d)	FDCBdd1E	S Z P C	... et l'adresse IY+d
RRA	1F	C	... et l'accu
RRC A	CB0F	S Z P C	Rotation à droite de l'accu
RRC B	CB08	S Z P C	... de B
RRC C	CB09	S Z P C	... de C
RRC D	CB0A	S Z P C	... de D
RRC E	CB0B	S Z P C	... de E
RRC H	CB0C	S Z P C	... de H
RRC L	CB0D	S Z P C	... de L
RRC (HL)	CB0E	S Z P C	... de l'adresse dans HL
RRC (IX+d)	DDCBdd0E	S Z P C	... de l'adresse IX+d
RRC (IY+d)	FDCBdd0E	S Z P C	... de l'adresse IY+d
RRCA	0F	C	... de l'accu
RRD	ED67	S Z P	Rotation décimale à droite
RST 00	C7		Appelle un ss-pg si 00
RST 08	CF		... si 08
RST 10	D7		... si 10
RST 18	DF		... si 18

RST 20	E7		... si 20
RST 28	EF		... si 28
RST 30	F7		... si 30
RST 38	FF		... si 38
SBC A,A	9F	S Z P C	Soustrait carry & A de A
SBC A,B	98	S Z P C	... B de l'accu
SBC A,C	99	S Z P C	... C de l'accu
SBC A,D	9A	S Z P C	... D de l'accu
SBC A,E	9B	S Z P C	... E de l'accu
SBC A,H	9C	S Z P C	... H de l'accu
SBC A,L	9D	S Z P C	... L de l'accu
SBC A,n	DEnn	S Z P C	... octet n de l'accu
SBC A,(HL)	9E	S Z P C	... adresse dans HL de A
SBC A,(IX+d)	DD9Edd	S Z P C	... adresse IX+d de A
SBC A,(IY+d)	FD9Edd	S Z P C	... adresse IY+d de A
SBC HL,BC	ED42	S Z P C	Soustrait carry & BC de HL
SBC HL,DE	ED52	S Z P C	... DE de HL
SBC HL,HL	ED62	S Z P C	... HL de HL
SBC HL,SP	ED72	S Z P C	... SP de HL
SCF	37	C=1	Met à 1 le bit de carry
SET 0,A	CBC7		Met à 1 bit 0 de l'accu
SET 0,B	CBC0		... de B
SET 0,C	CBC1		... de C
SET 0,D	CBC2		... de D
SET 0,E	CBC3		... de E
SET 0,H	CBC4		... de H
SET 0,L	CBC5		... de L
SET 0,(HL)	CBC6		... de l'adresse dans HL
SET 0,(IX+d)	DDCBddC6		... de l'adresse IX+d
SET 0,(IY+d)	FDCBddC6		... de l'adresse IY+d
SET 1,A	CBCF		Met à 1 bit 1 de l'accu

SET 1,B	CBC8	... de B
SET 1,C	CBC9	... de C
SET 1,D	CBCA	... de D
SET 1,E	CBCB	... de E
SET 1,H	CBCC	... de H
SET 1,L	CBCD	... de L
SET 1,(HL)	CBCE	... de l'adresse dans HL
SET 1,(IX+d)	DDCBddCE	... de l'adresse IX+d
SET 1,(IY+d)	FDCBddCE	... de l'adresse IY+d
SET 2,A	CBD7	Met à 1 bit 2 de l'accu
SET 2,B	CBD0	... de B
SET 2,C	CBD1	... de C
SET 2,D	CBD2	... de D
SET 2,E	CBD3	... de E
SET 2,H	CBD4	... de H
SET 2,L	CBD5	... de L
SET 2,(HL)	CBD6	... de l'adresse dans HL
SET 2,(IX+d)	DDCBddD6	... de l'adresse IX+d
SET 2,(IY+d)	FDCBddD6	... de l'adresse IY+d
SET 3,A	CBDF	Met à 1 bit 3 de l'accu
SET 3,B	CBD8	... de B
SET 3,C	CBD9	... de C
SET 3,D	CBDA	... de D
SET 3,E	CBDB	... de E
SET 3,H	CBDC	... de H
SET 3,L	CBDD	... de L
SET 3,(HL)	CBDE	... de l'adresse dans HL
SET 3,(IX+d)	DDCBddDE	... de l'adresse IX+d
SET 3,(IY+d)	FDCBddDE	... de l'adresse IY+d
SET 4,A	CBE7	Met à 1 bit 4 de l'accu
SET 4,B	CBE0	... de B

SET 4,C	CBE1	... de C
SET 4,D	CBE2	... de D
SET 4,E	CBE3	... de E
SET 4,H	CBE4	... de H
SET 4,L	CBE5	... de L
SET 4,(HL)	CBE6	... de l'adresse dans HL
SET 4,(IX+d)	DDCBddE6	... de l'adresse IX+d
SET 4,(IY+d)	FDCBddE6	... de l'adresse IY+d
SET 5,A	CBEF	Met à 1 bit 5 de l'accu
SET 5,B	CBE8	... de B
SET 5,C	CBE9	... de C
SET 5,D	CBEA	... de D
SET 5,E	CBEB	... de E
SET 5,H	CBEC	... de H
SET 5,L	CBED	... de L
SET 5,(HL)	CBEE	... de l'adresse dans HL
SET 5,(IX+d)	DDCBddEE	... de l'adresse IX+d
SET 5,(IY+d)	FDCBddEE	... de l'adresse IY+d
SET 6,A	CBF7	Met à 1 bit 6 de l'accu
SET 6,B	CBF0	... de B
SET 6,C	CBF1	... de C
SET 6,D	CBF2	... de D
SET 6,E	CBF3	... de E
SET 6,H	CBF4	... de H
SET 6,L	CBF5	... de L
SET 6,(HL)	CBF6	... de l'adresse dans HL
SET 6,(IX+d)	DDCBddF6	... de l'adresse IX+d
SET 6,(IY+d)	FDCBddF6	... de l'adresse IY+d
SET 7,A	CBFF	Met à 1 bit 7 de l'accu
SET 7,B	CBF8	... de B
SET 7,C	CBF9	... de C

SET 7,D	CBFA		... de D
SET 7,E	CBFB		... de E
SET 7,H	CBFC		... de H
SET 7,L	CBFD		... de L
SET 7,(HL)	CBFE		... de l'adresse dans HL
SET 7,(IX+d)	DDCBddFE		... de l'adresse IX+d
SET 7,(IY+d)	FDCBddFE		... de l'adresse IY+d
SLA A	CB27	S Z P C	Décale à gauche carry & accu
SLA B	CB20	S Z P C	... & B
SLA C	CB21	S Z P C	... & C
SLA D	CB22	S Z P C	... & D
SLA E	CB23	S Z P C	... & E
SLA H	CB24	S Z P C	... & H
SLA L	CB25	S Z P C	... & L
SLA (HL)	CB26	S Z P C	... & l'adresse dans HL
SLA (IX+d)	DDCBdd26	S Z P C	... & l'adresse IX+d
SLA (IY+d)	FDCBdd26	S Z P C	... & l'adresse IY+d
SRA A	CB2F	S Z P C	Décale à droite carry & accu
SRA B	CB28	S Z P C	... & B
SRA C	CB29	S Z P C	... & C
SRA D	CB2A	S Z P C	... & D
SRA E	CB2B	S Z P C	... & E
SRA H	CB2C	S Z P C	... & H
SRA L	CB2D	S Z P C	... & L
SRA (HL)	CB2E	S Z P C	... & l'adresse dans HL
SRA (IX+d)	DDCBdd2E	S Z P C	... & l'adresse IX+d
SRA (IY+d)	FDCBdd2E	S Z P C	... & l'adresse IY+d
SRL A	CB3F	S Z P C	Décale à droite carry & accu
SRL B	CB38	S Z P C	... & B
SRL C	CB39	S Z P C	... & C
SRL D	CB3A	S Z P C	... & D

SRL E	CB3B	S Z P C	... & E
SRL H	CB3C	S Z P C	... & H
SRL L	CB3D	S Z P C	... & L
SRL (HL)	CB3E	S Z P C	... & l'adresse dans HL
SRL (IX+d)	DDCBdd3E	S Z P C	... & l'adresse IX+d
SRL (IY+d)	FDCBdd3E	S Z P C	... & l'adresse IY+d
SUB A	97	S Z P C	Soustrait l'accu de l'accu
SUB B	90	S Z P C	... B de l'accu
SUB C	91	S Z P C	... C de l'accu
SUB D	92	S Z P C	... D de l'accu
SUB E	93	S Z P C	... E de l'accu
SUB H	94	S Z P C	... H de l'accu
SUB L	95	S Z P C	... L de l'accu
SUB n	D6nn	S Z P C	... octet n
SUB (HL)	96	S Z P C	... l'adresse dans HL
SUB (IX+d)	DD96dd	S Z P C	... l'adresse IX+d
SUB (IY+d)	FD96dd	S Z P C	... l'adresse IY+d
XOR A	AF	S Z P C=0	OU exclusif A avec A
XOR B	A8	S Z P C=0	... avec B
XOR C	A9	S Z P C=0	... avec C
XOR D	AA	S Z P C=0	... avec D
XOR E	AB	S Z P C=0	... avec E
XOR H	AC	S Z P C=0	... avec H
XOR L	AD	S Z P C=0	... avec L
XOR n	EEnn	S Z P C=0	... avec octet n
XOR (HL)	AE	S Z P C=0	... avec adresse dans HL
XOR (IX+d)	DDAEdd	S Z P C=0	... avec adresse IX+d
XOR (IY+d)	FDAEdd	S Z P C=0	... avec adresse IY+d

14. TRUCS ET FORMULES EN BASIC

Les trucs et formules qui suivent vous aideront à résoudre de nombreux petits problèmes qui peuvent se présenter lors de la programmation, ainsi que vous éviter des "errors" toujours désagréables. Nous commencerons par quelques formules mathématiques.

Vous avez peut-être remarqué que le CPC (comme les autres micro-ordinateurs d'ailleurs) ne connaît que le logarithme naturel (LOG) et le logarithme décimal (LOG10). Heureusement il existe une formule très simple permettant de calculer des logarithmes de base quelconque:

```
LOGn(x) = LOG(x)/LOG(n)
```

Il est également possible de calculer les fonctions inverses des fonctions trigonométriques. On peut ainsi établir ARCCOSINUS et ARCSINUS à partir de l'ARCTANGENTE (ATN):

```
ARCCOS(x) = -ATN(x/SQR(1-x*x))+PI/2
ARCSIN(x) = ATN(x/SQR(1-x*x))
```

Pour simplifier des fractions, il faut calculer le plus grand diviseur commun (PGCD) de deux nombres. Très souvent on emploie alors l'algorithme d'Euclide qui possède l'avantage de travailler très rapidement.
Les variables P et Q contiennent les deux nombres dont on veut calculer le PGCD. Le reste de la division entière P/Q est affecté à la variable R. Puis P obtient l'ancienne valeur de Q, et Q la valeur de R. Ce cycle est répété jusqu'à ce que R=0. P contient alors le PGCD

recherché.

Pour mieux comprendre les différentes étapes, voyons un exemple:

P	Q	R	
56	21	14	valeurs de départ
21	14	7	après 1er passage
14	7	0	après 2ème passage
7	0	0	P contient le PGCD

Le listing est ridiculement court:

```
10 p=nombre1: q=nombre2: r=1
20 WHILE r  0: r=p MOD p: p=q: q=r: WEND
```

Le R=1 est indispensable pour éviter que la condition en ligne 20 soit vérifiée avant même que la boucle n'aie été parcourue une seule fois.

Pour connaître le nombre de chiffres avant la virgule que comporte un nombre, vous pouvez employer la formule suivante:

```
S = INT(LOG10(ABS(x))+1)
```

Grâce à la fonction ABS il est également possible d'appliquer la formule aux nombres négatifs, LOG10 n'étant défini que pour les nombres positifs.

Voici maintenant une méthode pour générer un nombre aléatoire compris entre a et b (et pas entre 0 et b comme on en a l'habitude):

```
x = INT(RND(1)*(b-a+1)+a)
```

Un petit truc pour la conversion des nombres hexadécimaux et binaires. Lorsque l'on veut convertir, à l'aide de & et &X, des nombres supérieurs à &7FFF ou dont le bit de poids fort est à 1 (ce qui revient au même), on obtient un résultat négatif. Dans ce cas il suffit d'y additionner 65535 et le tour est joué.

Beaucoup de traitements de texte se distinguent par leur aptitude à pouvoir centrer une expression. Une simple formule BASIC peut produire l'effet recherché:

```
PRINT TAB ((longueur de ligne - LEN(texte$))/2);texte$
```

Longueur de ligne dépend du mode écran et texte$ contient la ligne à afficher.
Le dernier truc concerne les chaînes de caractères. Il s'agit d'une caractéristique désagréable de l'interpréteur du CPC (qu'il partage d'ailleurs avec d'autres micros). Lorsque l'on demande le code ASCII d'une chaîne vide (par exemple ASC("")), le CPC vous rétorque froidement un "improper argument". Voici comment l'éviter:

```
PRINT ASC(a$+CHR$(0))
```

Et tout rentre dans l'ordre.

ANNEXE

I. TABLEAU DES ADRESSES D'IMPLANTATION

Le tableau qui suit reproduit les fonctions de toutes les cases mémoires (connues). Naturellement certaines fonctions peuvent avoir des effets secondaires imprévus, c'est pourquoi les PEEK et les POKE dans certaines zones mémoire sont à employer avec circonspection. Néanmoins que cela ne vous empêche pas d'expérimenter, mais n'oubliez pas de sauvegarder auparavant d'éventuels programmes !

0000-3FFF	ROM contenant le système d'exploitation
0000-003F	copie de la ROM pour le bankswitching
0040-013F	tampon clavier et zone de travail
0170-AB7F	mémoire pour programmes BASIC
3800-3FFF	ROM contenant la police de caractère
AB80-ABFF	caractères définissables par l'utilisateur
AC00	flag pour comprimer les lignes BASIC
AC01-AC03	saut d'extension pour le mode READY
AC04-AC06	// // // le traitement des erreurs
AC07-AC09	// // // l'exécution d'instructions
AC0A-AC0C	// // // le calcul de fonctions
AC0D-AC0F	// // // la lecture d'une constante
AC10-AC12	// // // l'entrée d'une ligne BASIC
AC13-AC15	// // // LIST
AC16-AC18	// // // conversion de nombres
AC19-AC1B	// // // les opérateurs

AC1C	flag pour AUTO
AC1D-AC1E	numéro de ligne AUTO
AC1F-AC20	largeur du pas pour AUTO
AC21	dernier numéro de "stream"
AC22	canal d'entrée
AC23	dernière position de l'imprimante
AC24	WIDTH
AC25	dernière position sur cassette
AC26	flag pour FOR-NEXT
AC27-AC2B	mémoire auxiliaire pour FOR-NEXT
AC2C-AC2D	adresse pour NEXT
AC2E-AC2F	// // WEND
AC34-AC35	// // ON-BREAK
AC38-AC43	séquence de sons 0
AC44-AC4F	// // 1
AC50-AC5B	// // 2
AC5C-AC6D	domaine pour interruption 0
AC6E-AC7F	// // // 1
AC80-AC91	// // // 2
AC92-ACA3	// // // 3
ACA4-ADA3	mémoire tampon
ADA6-ADA7	adresse des ERRORs
ADA8-ADA9	pointeur BASIC après un ERROR
ADAA	numéro de l'erreur
ADAB-ADAC	pointeur BASIC après une interruption
ADAD-ADAE	adresse de la ligne interrompue
ADAF-ADB0	adresse pour ON-ERROR
ADB1	flag: ON-ERROR est actif
ADB2	état du canal
ADB3	ENT
ADB4	ENV

ADB5-ADB6 période

ADB7 période du bruit blanc

ADB8 volume

ADB9-ADBA longueur

ADBB-ADBC ENV et ENT

ADCB-ADCF mémoire auxiliaire p. calcul en virgule flottante

ADD0-AE03 tableau de variables scalaires

AE04-AE05 tableau FN

AE06-AE0B tableau "array"

AE0C-AE25 type des variables prédéfinies A à Z

AE2D caractère de séparation pour INPUT

AE2E-AE2F adresse pour READ

AE30-AE31 adresse pour DATA

AE32-AE33 adresse du pointeur de pile-BASIC

AE34-AE35 adresse de l'instruction en cours

AE36-AE37 adresse de la ligne programme en cours

AE38 flag pour TRACE

AE3F-AE40 adresse de départ pour LOAD

AE41 flag pour CHAIN MERGE

AE42 type du fichier

AE43-AE44 longueur du fichier

AE45 flag de protection de programme

AE46-AE78 zone de travail pour conversion en ASCII

AE72-AE73 adresse de CALL

AE74 mode bankswitching pour CALL

AE75-AE76 registre HL pour CALL

AE77-AE78 registre SP pour CALL

AE79 grandeur de la tabulation

AE7B-AE7C pointeur HIMEM

AE7D-AE7E pointeur sur fin de la RAM disponible

AE7F-AE80 // // début de la RAM disponible

```
AE81-AE82  //     //   //   du programme BASIC
AE83-AE84  //     //  fin du programme BASIC
AE85-AE86  //     //  début des variables
AE87-AE88  //     //  début des tableaux
AE89-AE8A  //     //  fin des tableaux
AE8B-B08A pile pour BASIC (FOR, GOSUB etc.)
B08B-B08C pointeur de pile BASIC
B08D-B08E pointeur sur début des strings
B08F-B090 pointeur sur fin des strings
B09A-B09B pointeur sur pile string
B09C-B0B9 pile string
B0BA-B0BC stringdescriptor
B0C1      type de la variable
B0C2-B0C3 diverses adresses
B100-B1AB zone de travail du système d'exploitation
B1C8      mode de l'écran actuel
B1CA      offset de la mémoire écran
B1CB      adresse de la mémoire écran
B1CC-B1D6 utilisations diverses
B1D7-B1D8 vitesse de clignotement
B1D9-B20B divers registres pour couleurs changeantes
B20C      fenêtre actuelle
B20D-B276 paramètres pour les fenêtres
B285-B286 position du curseur (ligne,colonne)
B287-B28B divers registres pour les fenêtres
B28C-B327  //      //      // l'écran
B328-B329 ORIGIN-X
B32A-B32B ORIGIN-Y
B32C-B4DD divers registres pour le graphisme
B4DE-B550  //      //      // la lecture du clavier
B551-B7FF  //      //      // le son
```

B800-B8DC	// // // la cassette
B807-B816	nom du fichier INPUT
B84C-B85B	nom du fichier OUTPUT
B8D1	vitesse d'écriture
B8DD	flag pour l'insertion
B8E4-B8E7	nombre aléatoire
B8E8-B8F6	mémoire auxiliaire p. calcul en virgule flottante
B8F7	flag pour DEG/RAD
BF00-BFFF	pile pour le Z-80
C000-FFFF	RAM vidéo
C000-FFFF	ROM interpréteur
C000-FFFF	ROM d'extension

II. INDEX

C

D

H

I

L

M

N

O

P

R

S





LA BIBLE DU PROGRAMMEUR DE L'AMSTRAD CPC

LA BIBLE DE L'AMSTRAD CPC est une aide indispensable pour les programmeurs en BASIC et le MUST absolu pour les programmeurs en assembleur. Cet ouvrage de référence qui révèle vraiment tous les secrets du CPC, est le fruit d'un travail minutieux de plusieurs mois.

Contenu :

- organisation de la mémoire - le processeur - particularité du Z 80, du CPC - GATE ARRAY - le contrôleur vidéo - la ROM vidéo - le CHIP sonore - les interfaces - les systèmes d'exploitation - utilisation des routines avec l'exemple du HARD COPY - le générateur de caractères - l'interpréteur BASIC - BASIC et langage machine - le listing de la ROM - etc.

Prix 249 F TTC   Ref ML 122

LE BASIC AU BOUT DES DOIGTS CPC 464

Ce livre est une introduction complète et didactique au BASIC du micro-ordinateur AMSTRAD CPC 464.

Il permet d'apprendre rapidement et facilement la programmation (instructions BASIC, analyses des problèmes, algorythmes complexes...).

Principaux thèmes abordés : • Les bases de la programmation • Bit, Octet, ASCII • Instructions du BASIC • Organigrammes • Les fenêtres • Programmes BASIC plus poussés • Le programme et menus.

Comprenant de nombreux exemples, ce livre vous assure un apprentissage simple et efficace du BASIC CPC 464.

Prix : 149 F TTC.   Réf. : ML 118

AMSTRAD CPC 464. TRUCS ET ASTUCES

De nombreux trucs et astuces pour le CPC 464. La structure hardware, du système d'exploitation, des tokens BASIC, du dessin avec le joystick, des applications de la technique des fenêtres, et d'un grand nombre de programmes intéressants tels qu'une gestion de fichier complète, d'un éditeur de son, d'un générateur de caractères commodes jusqu'aux listings complets de jeux passionnants.

Prix : 149 F TTC   Réf. : ML 112

LES JEUX D'AVENTURES. COMMENT LES PROGRAMMER.

Voici la clé du monde de l'aventure. Ce livre fournit un système d'aventures complet, avec éditeur, interpréteur, routines utilitaires et fichiers de jeux. Ainsi qu'un générateur d'aventures pour programmer vous-même facilement vos jeux d'aventures. Avec, bien sûr, des programmes tout prêts à être tapés.

Prix : 129 F TTC   Réf. : ML 121

PEEKS ET POKES DU CPC

Les PEEKS, POKES et CALLS sont une introduction aisée à la compréhension du système d'Exploitation et du Langage Machine de l'AMSTRAD CPC. De nombreuses et intéressantes possibilités de programmation et d'applications sont abordées dans ce livre. Quelques extraits : - Configuration "HARDWARE" du CPC - Système d'exploitation et interpréteur. - PEEK et POKE - CALL. - Calcul binaire. - Protection de la mémoire.

Prix : 99 F TTC
Réf. : ML 126.

LE LANGAGE MACHINE POUR L'AMSTRAD CPC

"Le langage machine pour l'AMSTRAD CPC" est fait pour tous ceux qui considère que le BASIC n'est plus ni assez puissant ni assez rapide. Des bases de la programmation en langage machine au mode de travail du processeur Z 80 en passant par une description précise de ses instructions ainsi que l'utilisation des routines systèmes.

Tout est expliqué complètement et avec de nombreux exemples.

Le livre contient des programmes complets : un assembleur, un désassembleur et un moniteur. Grâce à ce livre, le langage machine n'aura plus de secret pour vous.

Prix : 129 F TTC   Ref. ML 123

GRAPHISMES ET SONS DU CPC

L'AMSTRAD CPC dispose de capacités graphiques et sonores exceptionnelles. Ce livre en montre l'utilisation à l'aide de nombreux programmes intéressants et utilitaires. Contenu : - base de programmation graphique - éditeur de police de caractères - "sprites", "shapes", et chaînes - représentations multi-couleurs - calcul des coordonnées - rotations, mouvements - représentations graphiques de fonctions en 3D - D.A.O. (dessin assisté par ordinateur) - synthétiseur - mini-orgue - enveloppes de son, et beaucoup d'autres choses...

Prix : 129 F TTC   Réf. : ML 124.

AMSTRAD OUVRE-TOI

Le bon départ avec le CPC 464 ! Ce livre vous apporte les principales informations sur l'utilisation, les possibilités de connexions du CPC 464 et les rudiments nécessaires pour développer vos propres programmes. C'est le livre idéal pour tous ceux qui veulent pénétrer dans l'univers des micro-ordinateurs avec le CPC 464.

Prix : 99 F TTC   Réf. : ML 120

AMSTRAD AUTOFORMATION À L'ASSEMBLEUR EN FRANÇAIS

Contient un livre et un logiciel.

Le livre : Cet ouvrage introduit le débutant à la programmation du Z80 grâce à la méthode du Dr WATSON qui selon les critiques vaut son pesant d'or ! Aucune connaissance préalable n'est requise et le but du livre est d'assurer au novice un succès total. A la fin du livre les instructions du Z80 sont expliquées en détail. De nombreux exemples illustrent les différentes étapes du cours alors que des exercices (les solutions sont fournies) testent la compréhension.

Le logiciel :

Un assembleur Z80 complet est livré sur cassette et comprend : • Etiquettes Symboliques • Directives d'Assemblage • Chargement/Sauvegarde • Copie Écran • INSERT/DELET. L'assembleur permet d'écrire des programmes facilement en langage d'assemblage puis les transforme en code machine (langage machine). Pour vous aider à comprendre les rotations mathématiques utilisées, une démonstration de l'utilisation des nombres binaires et hexadécimaux est fournie. Un programme utilisant les commandes graphiques additionnelles décrites dans le livre est également fourni.

Prix : 198 F TTC   Réf. : ML 126

PROGRAMMES BASIC POUR LE CPC 464

Alimentez votre CPC 464.

Ce livre contient de super programmes, notamment un désassembleur, un éditeur graphique, un éditeur de texte... tous les programmes sont prêts à être tapés et abondamment commentés.

* Prix : 129 F TTC   Réf. : ML 119

NUMÉRO 1 - MAI 85 - Prix 15 F. Belgique 120 FB. Suisse 4 FS.

MICRO INFO

LES PIRATES, CES GÉNIES MÉCONNUS !

À AFFICHER

LES PEEKS ET LES POKES DU COMMODORE 64

AVANT-PREMIÈRE

LE COMMODORE PC

DU NOUVEAU POUR L'AMSTRAD CPC 464

DOSSIER : TOUT, TOUT, TOUT SUR LE COMMODORE 128

EXCLUSIF ! L'ATARI ST n'est plus qu'à 800 KM.

CONCOURS AMSTRAD

50 PRIX À GAGNER

Le programme le plus drôle !

Cher Client,

Nous avons le plaisir de vous annoncer le lancement de notre revue « MICRO INFO ».

Ce magazine comprend des trucs, des astuces, des conseils, des programmes, des concours, des enquêtes et études.

Dans le premier numéro (84 pages) vous trouverez notamment une enquête sur les pirates, un dossier sur le nouveau Commodore 128 et sur l'unité de disquettes AMSTRAD, un poster des Peeks et Pokes du COMMODORE 64...

Pour recevoir le numéro 1 de MICRO INFO retournez-nous, par chèque ou CCP, la somme de 15 FF + 5 FF de port c'est-à-dire un total de 20 FF à

MICRO APPLICATION
13, rue Sainte-Cécile
75009 PARIS

NOM : ....

Prénom : .... *Signature :*

Adresse : ....

Ville : .... □ Chèque □ CCP

Code Postal : .... à l'ordre de
MICRO APPLICATION

Date

Nous aimerions que l'information circule dans les deux sens !
Cette page vous est réservée.

NOM :	Matériel utilisé :
Prénom :	Date d'Achat :
Adresse :	Extension/Périphérique :
Code Postal :	Logiciel préféré :
Age :	
Sexe :	

Etes-vous satisfait des logiciels existant ? Oui Non

Si oui, lesquels ?

Si non, quels sont les logiciels que vous aimeriez trouver ?

Que pensez-vous des logiciels MICRO APPLICATION ?

Que pensez-vous des livres MICRO APPLICATION ?

Que pensez-vous de la revue ?

Votre rubrique personnelle :

ABONNEMENT

Attention Micro Info n'étant tiré qu'à 10 000 exemplaires.
Réservez dès à présent le numéro spécial Rentrée 85 plus les 3 prochains numéros pour la somme de 60 FF.

Règlement par chèque bancaire ou CCP uniquement.

Bulletin d'abonnement :

NOM :
Prénom :
Adresse :

Code Postal : Ville :

Nos petites Annonces gratuites seront réservées en priorité aux abonnés.

Achevé d'imprimer en octobre 1985
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : octobre 1985
N° d'imprimeur : 509090

Achevé d'imprimer en juin 1985
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : juin 1985
N° d'imprimeur : 506058

IMPRIMÉ EN FRANCE

[redacted]

Les PEEKS, POKES et CALLS sont une introduction aisée à la compréhension du système d'Exploitation et du Langage Machine de l'AMSTRAD CPC. De nombreuses et intéressantes possibilités de programmation et d'applications sont abordées dans ce livre.

Quelques extraits :

- Configuration "HARDWARE" du CPC.
- Système d'exploitation et interpréteur.
- PEEK et POKE - CALL.
- Calcul binaire.
- Protection de la mémoire.
- Bankswitching - Lire la ROM.
- Mémoire écran - Graphisme - Scrolling.
- Interruptions en BASIC.
- Représentation en mémoire de lignes BASIC.
- Garbage collection.
- Fonctionnement du micro-processeur Z 80.
- Possibilités d'adressage.
- Nombreuses routines en Langage Machine.

Avec ce livre, vous comprendrez aisément le fonctionnement du CPC et profiterez pleinement de ses extraordinaires capacités.

ISBN : 2-86899-017-7

Réf. : ML 126

Prix : 99 FF



AMSTRAD
PEEKS ET POKES
DU CPC
9
MA



Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE

ACME

https://acpc.me/